NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

**MANAGEMENT SYSTEM
FOR
HETEROGENEOUS NETWORKS
SECURITY SERVICES**

by

Roger E. Wright

June 1998

Thesis Advisor:	Cynthia E. Irvine
Second Reader:	Debra Hensgen

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1998	Master's Thesis

4. TITLE AND SUBTITLE
MANAGEMENT SYSTEM FOR HETEROGENEOUS NETWORKS SECURITY SERVICES

5. FUNDING NUMBERS

6. AUTHOR(S)
Wright, Roger E.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Military C4I facilities form an enormous network of distributed, heterogeneous computers. Operating these computers such that commanders can exploit their computing power effectively requires a resource management system. Management System for Heterogeneous Networks (MSHN) is a program under development specifically designed to address this need. Security for distributed computing systems is of particular importance to the Department of Defense. Previously developed resource management systems have largely neglected the issue of security. This thesis proposes a security architecture through which MSHN can achieve its goal of providing optimal usage of compute resources while simultaneously providing security commensurate with the software and data processed. A demonstration of the security framework was created using Intel Corporation's Common Data Security Architecture (CDSA). CDSA provided the cryptographic mechanisms required to build the security framework.

14. SUBJECT TERMS
MSHN, Distributed Computing, Security Mechanisms, Common Data Security Architecture, Virtual Heterogeneous Machine

15. NUMBER OF PAGES
283

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

MANAGEMENT SYSTEM FOR HETEROGENEOUS NETWORKS SECURITY SERVICES

Roger E. Wright
Captain, United States Army
B.S., Loyola University of Chicago, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY
(Command, Control and Communications)

from the

NAVAL POSTGRADUATE SCHOOL
June 1998

Author: ______________________
Roger E. Wright

Approved by: ______________________
Cynthia E. Irvine, Thesis Advisor

____
Debra Hensgen, Second Reader

____
Dan Boger, Chair
C4I Academic Group

ABSTRACT

Military C4I facilities form an enormous network of distributed, heterogeneous computers. Operating these computers such that commanders can exploit their computing power effectively requires a resource management system. Management System for Heterogeneous Networks (MSHN) is a program under development specifically designed to address this need.

Security for distributed computing systems is of particular importance to the Department of Defense. Previously developed resource management systems have largely neglected the issue of security. This thesis proposes a security architecture through which MSHN can achieve its goal of providing optimal usage of computing resources while simultaneously providing security commensurate with the software and data processed.

A demonstration of the security framework was created using Intel Corporation's Common Data Security Architecture (CDSA). CDSA provided the cryptographic mechanisms required to build the security framework.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

The author gratefully acknowledges the advice and support of Dr. Cynthia Irvine and David Shifflett. Their assistance and guidance proved to be essential in the completion of this thesis. Dr. D. Hensgen provided essential insights regarding the architecture of the MSHN VHM. I would also like to thank Dan Warren for sparking my interest in computer security. Finally, a heartfelt thanks is offered to my wife and children, who supported me throughout this effort.

EXECUTIVE SUMMARY

Military C4I facilities form an enormous network of distributed, heterogeneous computers. Operating these computers such that commanders can exploit their computing power effectively requires a resource management system. Management System for Heterogeneous Networks (MSHN) is a program under development specifically designed to address this need.

Security for distributed computing systems is of particular importance to the Department of Defense. Previously developed resource management systems have largely neglected the issue of security. This thesis proposes a security architecture through which MSHN can achieve its goal of providing optimal usage of computing resources while simultaneously providing security commensurate with the software and data processed.

A demonstration of the security framework was created using Intel Corporation's Common Data Security Architecture (CDSA). CDSA provided the cryptographic mechanisms required to build the security framework. CDSA is a layered architecture that presents a common interface through which application programmers and cryptographic hardware/software vendors can develop their respective products independently, yet be assured they will work together properly when integrated.

CDSA is fundamentally dependent upon a Public/Private Key Infrastructure (PKI). CDSA assumes that user identities will be encoded in the form of digital certificates. Digital signature and encryption mechanisms provided by CDSA form the basis for the security architecture proposed for MSHN.

The security framework for MSHN has resulted in the specification of a MSHN security layer which defines the interface through which MSHN components access security mechanisms. The MSHN security layer provides services for encryption, decryption, digital signatures and verification, certificate creation, verification and revocation, and other security-relevant functions.

The security-enhanced MSHN architecture uses the security framework to guarantee integrity and confidentiality of communications between the user, MSHN components, and the computing resources running user applications.

A proof of concept demonstration was implemented. It was written in C++ for execution on three Windows NT personal computers connected via a local area network. The demonstration program shows that we can secure MSHN communications and that CDSA is a viable option as a cryptographic applications programming interface.

I. INTRODUCTION

A. PURPOSE

The information technology community is in the midst of a paradigm shift. Until recently, the processing of highly compute-intensive applications was performed by isolated supercomputing platforms. It is now possible for such applications to be executed by a network of computers bound together by an overarching framework that presents the user with the appearance of one powerful, virtual heterogeneous machine (VHM).

Efforts are currently underway to define and implement a management system for user applications running in the VHM. Management System for Heterogeneous Networks (MSHN) is an emerging resource management system whose primary function is to accept user jobs, and determine what jobs should be executed on which machines throughout the VHM and when.

The purpose of this thesis is to discuss and analyze the security requirements of an application program running in a distributed, heterogeneous, networked environment. In particular, a security architecture suitable for Management System for Heterogeneous Networks (MSHN) will be proposed. Additionally, this thesis investigates the

possibility of implementing the proposed security architecture via commercial off-the-shelf security software.

Intel Corporation has recognized the need for a security infrastructure that is applicable to a distributed, multi-platform computing environment. The result of Intel's research is the publication of a security infrastructure termed Common Data Security Architecture (CDSA). The use of CDSA as a means of providing a suitable level of trust has been investigated. This thesis will analyze MSHN's security needs and recommend a solution via CDSA. Finally, a prototype demonstration of the security architecture is presented.

B. MOTIVATION

Significant challenges exist in securing any computing system, however, the usual problems are exacerbated when a process is distributed across numerous computing environments. Previously developed resource management systems have largely neglected the issue of security. Existing systems lack scalable mechanisms for authentication and privacy [Ref. 1]. The current prototype of MSHN does not provide protection against the threat of unauthorized disclosure or modification of the user's data.

Consider, for example, a user application which simulates the operation of a newly designed jet engine. This sort of simulation, depending on the fidelity of the

model, may require the enormous consumption of computational resources. Thus, such an application may be an ideal candidate for execution via MSHN. However, if the simulation results are not properly protected, their unauthorized disclosure could provide competitors with an unfair advantage.

Similarly, if the engine were destined for a military aircraft, compromise of the simulation results could provide adversaries with information that might imply countermeasure techniques. Or worse, if the adversary could modify the simulation results without detection, they might cause the design to be flawed. Clearly, the users of MSHN will demand protection of their data, or they will be unwilling to request its services.

C. ORGANIZATION

1. Distributed Computing

This chapter will discuss the benefits of distributed computing and describe an example application of a military command and control decision support tool that would be suited to operating in a distributed computing environment.

2. Management System for Heterogeneous Networks

This chapter will describe the status of the MSHN architecture, which is currently under development. It provides a high level view of the processing and

communications that must take place between the components of MSHN.

3. Essentials of Secure Systems

This chapter discusses the fundamentals of computer security. It describes modern security techniques and mechanisms.

4. CDSA

This chapter describes Intel's Common Data Security Architecture (CDSA). Along with an overview of the CDSA design, this chapter discusses the advantages and disadvantages of Intel's specification.

5. Security Enhanced MSHN Architecture

This chapter will describe an augmented architecture for MSHN that utilizes underlying capabilities to provide limited security protection. The security-enhanced architecture originates from the description of MSHN in Chapter III, modified to take advantage of the security mechanisms noted in Chapter IV. A proof of concept demonstration program, written using CDSA, will be described. It implements a primitive set of security mechanisms for the security enhanced MSHN architecture.

6. Recommendations and Conclusions

This chapter recommends areas of additional research and concludes with a summary analysis of MSHN security.

II. DISTRIBUTED COMPUTING

For many years the computer industry has been progressing from centralized computing (mainframes) to decentralized, distributed computing operations. Earlier generations of computer users were forced to rely upon centralized processing because of the computational power required, the lack of communications network capacity, and because everyone could not afford to purchase their own computers. This situation no longer exists.

The explosive growth in personal computer and workstation usage has resulted from their ever-increasing processing capabilities, continued decrease in acquisition costs, and substantial user friendliness improvements provided via graphical user interfaces. Personal computers and workstations have empowered users to control the processing of their data in a more direct manner.

Moreover, distributed computing has become the method of choice not only because of the personal computer evolution, but also because of the connectivity of personal computers to more powerful computers via local and wide area networks. The ability to share data resources among multitudes of disparate users is a tremendous benefit to all. The Department of Defense (DoD), like any other enterprise, realizes the significance of information technology as a key component, critical to the success of

its mission. The philosophy for the integration of information technology into the DoD is described below.

A. JOINT VISION 2010

The Chairman of the Joint Chiefs of Staff has outlined his future vision of twentieth century warfare in a document titled *Joint Vision 2010* [Ref. 2]. Figure 1 captures the concepts of JV 2010 in one diagram.

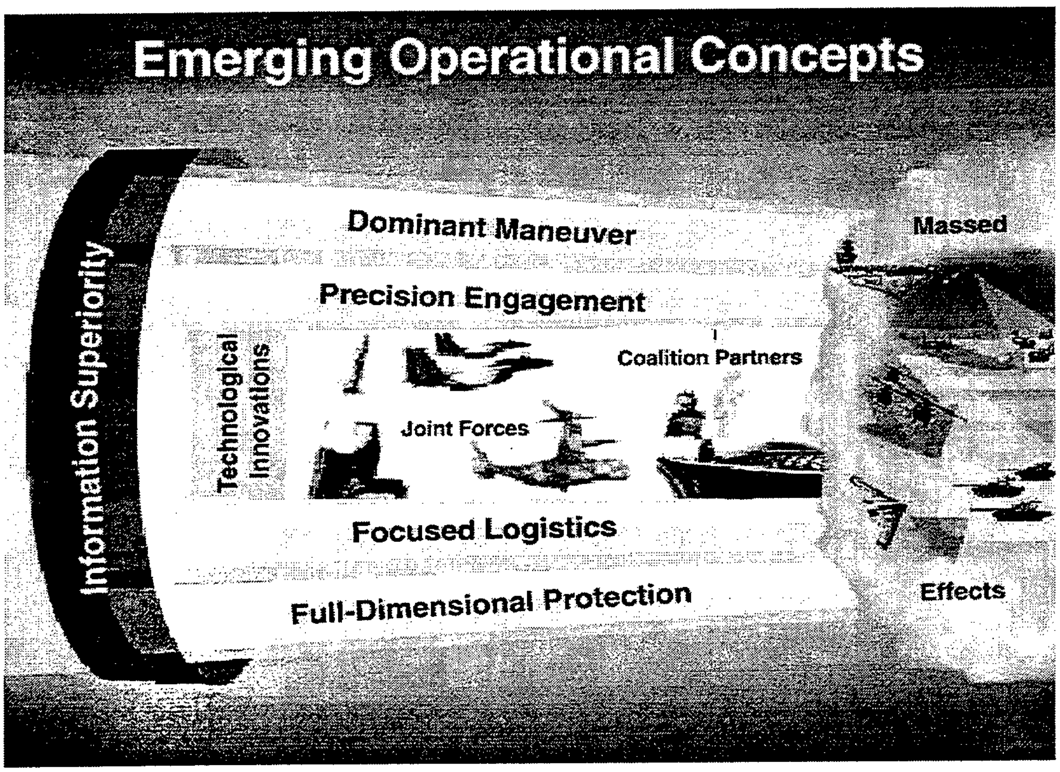


Figure 1 Tenets of JV 2010 [Ref. 2]

JV 2010 is based upon the tenets of dominant maneuver, precision engagement, focused logistics, and full-dimensional protection. Note that the four tenets are all encompassed by *information superiority*. In an era of

diminishing force structure, the success of our nation's military defense is tied to its ability to do more with less. Information superiority is the force multiplier that enables a smaller, more agile, technologically superior force to succeed in battle. The doctrine that outlines how the Department of Defense will achieve information superiority is published in Joint Pub 6 [Ref. 3].

B. C4I FOR THE WARRIOR: JOINT PUB 6

C4I represents command, control, communications, computers, and intelligence. C4I encompasses the procedures that a commander employs to direct his forces, as well as the policies to be used to communicate information between them. *C4I For the Warrior* is the vision for future command and control systems. In this case, the *Warrior* refers to the warfighting commander. In order to accomplish his mission, the warrior needs a fused, current, and accurate representation of the battlespace along with the ability to coordinate with, respond to, and order all his forces. The U.S. Marine Corps concisely defines the importance of command and control in the following quote from its doctrine:

> War is a process that pits the opposing wills of two commanders against each other. Great victories of military forces are often attributed to superior firepower, mobility, or logistics. In actuality, it often is the commander who makes good decisions and executes these decisions at a superior tempo who leads his forces to victory.

> Therefore, victory demands that commanders effectively link decision making to execution through the concept of command and control. Warfare will continue to evolve and command and control processes, organization, and supporting systems will continue to change, but the basic concept of command and control will remain the key to the decisive application of combat power. More than ever before, a command and control system is crucial to success and must support shorter decision cycles and instantaneous flexibility across vast distances of time and space. [Ref. 3]

Clearly, if we are to achieve information dominance, we need automated tools to help us develop plans faster than our adversary. Command of joint forces in war is an intense, competitive and stressful process. The joint force commander is not only faced with making life and death decisions in complex situations but must do this, in limited time, in an environment of uncertainty. Command is as much a problem of information management as it is of carrying out difficult and complex warfighting tasks.

Command, control, communications, computers and Intelligence (C4I) systems supporting US military forces must have the capability to rapidly ***adapt*** to the demands of the commanders who use them, as well as the environment in which they are used. They must make available the information that is important; provide it where needed; and ensure that it gets there not only in a timely manner, but also in a format that is usable by the receiver. The Joint Chiefs of Staff summarize the goal of C4I systems as follows: "The fundamental objective of C4I systems is to get

the critical and relevant information to the right place in time to allow forces to seize on opportunity and meet the objectives across the range of military operations." [Ref. 3].

Decision Support Systems are at the heart of the commander's planning process. Computer driven simulation programs can be used to assess the likely effectiveness of a given war plan before any casualties are incurred. The results of the simulation runs can then be used to adjust or modify plans until the commander is satisfied with their predicted results. Decision support through modeling and simulation can provide a significant advantage to the warfighter. An example of their use is described next.

C. C3ISIM: A CASE STUDY

C3ISIM is a simulation model developed as a tool to study Command, Control, Communications, and Intelligence related issues [Ref. 4]. The following analysis discusses the nature and purpose of C3ISIM, to include its usage in Operation Desert Shield/Storm, as well as an assessment of the value of its contribution in that engagement.

The analysts who employed C3ISIM in support of Operation Desert Shield initially attempted to use the model to assist in the analysis of air defense networks currently in place in Saudi Arabia and Iraq. While researching and collecting data on air defense equipment, the mission

changed. As the air campaign planners developed their strategy, the C3ISIM analysts were tasked to perform a detailed study of the first few hours of the attack plan. This new task was added because of C3ISIM's ability to simulate both fighter to fighter engagements and SAM (Surface to Air Missile) to fighter engagements. Additionally, it shows via computer display, a high resolution graphic display of the simulated battle as it unfolds. This unique feature, along with the raw attrition numbers generated by the program, enabled the air campaign planners to maximize their measure of force effectiveness (MOFE), which in this case, was to minimize attrition of friendly aircraft.

C3ISIM proved to be a valuable tool in the development of the Desert Storm Air Campaign. C3ISIM enabled planners to "play" their campaign plan on the computer. From the simulation they were able to perceive potential attrition "hot spots" as well as vulnerabilities in the Iraqi air defense system. Planners could use this information and tailor their war plan to avoid the enemy's strengths and exploit its weaknesses. As a decision aid, the information generated by C3ISIM may have saved the lives and expensive aircraft of U.S. pilots. Of course, it must be understood that it was only one of many tools used to aid in the decision making process. Other analysis methods and models

were investigated and used for comparison. Ultimately, C3ISIM was perceived as a valid model by the users because it passed their "gut check." Also noteworthy is what C3ISIM did not do. It did not develop the air campaign, nor did it make the process any easier or faster. What it did do was help the planners produce a better overall product. And from this perspective, C3ISIM was completely successful.

Models, by definition, are an abstraction of reality. Analysts, as well as the user community, must remember that a model, no matter how detailed, will never completely simulate all aspects of battle. The level of detail provided by C3ISIM was adequate for its intended use. However, there will always be a conflict between detail (realism) and speed of model execution.

C3ISIM was only marginally acceptable in terms of timeliness. Once the air campaign began, C3ISIM was no longer a viable tool because each simulation run took too long to execute. The C3ISIM analysts noted:

> The initial runs we made covered the first three hours of real time, but required almost eight hours to execute on the computer. We were able to conduct two executions per day at that rate. The number of times we ran the model with different 'rolls-of-the-dice' was certainly constrained by how quickly we needed the answer to the attrition question. [Ref. 4]

Like any other computer program, its outputs are only as good as the inputs. At this time the air tasking order was extremely volatile. Changes to the air picture occurred

on a continual basis. C3ISIM was slow, and since it was a Monte Carlo model, it required numerous executions, using the same data set, before trends could be reliably detected. Therefore, once the air plan had been established and started, C3ISIM could not produce attrition estimates in time for the execution of a mission. Throughout Desert Storm, the mission of C3ISIM turned to that of trying to determine the cause of downed aircraft, rather than attempting to predict future losses.

Nonetheless, C3ISIM demonstrated its applicability and suitability as a decision tool for combat planners. By analyzing and simulating an extremely complex problem, C3ISIM not only confirmed the planner's "gut feeling," but also provided insights that may have otherwise gone undiscovered.

C3ISIM is one of a number of complex decision support tools available to a commander. The compute power required to process these programs grows unbounded as the user demands more fidelity and realism from the software. During Desert Shield, C3ISIM was run on a Silicon Graphics 240GTX workstation. Consider the advantage that a network of cooperative computers would have provided if it had been available at the time. Simulation runs could have executed concurrently on dozens if not hundreds of powerful workstations. Simulation results would have been returned

in time for planners to adjust their battle plan. Thus, the usefulness of the C3ISIM model could have extended into the tactical battle planning process.

D. SUMMARY

Decision support systems and the explosive growth of networked computing have irrevocably changed the paradigm by which computers of all types are employed. Long gone are the days of mainframe monsters tied to dumb terminals. These changes have affected the manner in which the military and commercial industries will employ information technology assets. Joint Vision 2010 requires information superiority for success in battle. Distributed systems are a key enabler to that objective. It is imperative that the Department of Defense take advantage of distributed computing, for our adversaries surely will.

III. MANAGEMENT SYSTEM FOR HETEROGENEOUS NETWORKS (MSHN)

A. PURPOSE

Modern computer networks have grown in size and complexity as fast as the technology will allow. They also span greater distances and include machines of varying types. This expansion has resulted in the need to effectively manage large heterogeneous computer networks, to deliver good performance to all users, regardless of their individual measure of quality of service. In response to this need, a team of computer science experts, funded by the Defense Advanced Research Projects Agency (DARPA), is currently developing a resource management system named MSHN[1].

The goal of MSHN is to provide a computing environment that delivers each user's specified quality of service, subject to the available resources, the user's individual priorities, and the preference of each user for different forms of the requested data. Given a set of jobs, MSHN will determine where and when to run each job. MSHN evolved from SmartNet, which was a heterogeneous framework for minimizing the time at which the last job of a set of computationally intensive jobs finishes on a suite of heterogeneous

---

[1] Pronounced "mission".

computing resources [Ref. 5]. SmartNet treats the set of compute resources available as one virtual heterogeneous machine (VHM). SmartNet achieves superior performance by determining scheduling solutions based upon knowledge of the VHM and job characteristics. MSHN differs from SmartNet in several ways: (1) it strives to support Input/Output intensive and real-time jobs in addition to compute-intensive jobs; (2) it accounts for the fact that many different resources may be needed, not just a central processing unit, to execute a job; and (3) it manages *adaptive* applications.

One of the key improvements of MSHN over traditional resource management systems is that MSHN is intended to support adaptive applications. Adaptive applications are those which can produce results using either a variety of algorithms or in a variety of forms. MSHN uses knowledge of these various forms to choose the appropriate one for the given operating environment. MSHN is intended to help achieve the goals set forth in Joint Vision 2010, and C4I For The Warrior, by allowing a commander to maximize the utility of his computer network, particularly in a volatile wartime environment.

In addition to improving the performance of other applications, MSHN is intended to expand the usefulness of compute intensive, batch processed jobs such as C3ISIM. Had

MSHN been available during the Gulf War, it may have been possible to use C3ISIM's simulation results throughout the air campaign, as opposed to only prior to the start of the battle.

B. MSHN PROPOSED ARCHITECTURE

The MSHN architecture is still under development. The description that follows is based upon the architecture design as of April 30, 1998. While the ultimate MSHN architecture may differ slightly, the main components and concepts are expected to remain unchanged.

MSHN will consist of a client-server architecture. It will be composed of at least the following components:

- Client Library
- Scheduling Server
- Resource Requirement Database
- Resource Status Server

An abstract description of each of the components is provided below, and portrayed in Figure 2. Although these components are shown together, they may in fact reside on separate machines. There will usually be many different client applications, linked with the client libraries running at any given time. Additionally, it is conceivable that some of the components may be replicated.

1. Client Library

The client library is linked with both adaptive and non-adaptive applications. It provides a transparent interface to all of the MSHN services [Ref. 6]. The client library performs several functions. It intercepts system calls to (1) monitor end-to-end performance, (2) record resource requirements, and (3) forward requests to start another process, when appropriate, to the Scheduling Server. It forwards the end-to-end performance measurements to the Resource Status Server, and the recorded resource requirements to the Resource Requirements Database. The final implementation of MSHN may forward the performance measurements and resource requirements through a local proxy. Finally, it will also intercept calls from the Scheduling Server which indicate that a different form of the application, other than the one that is currently running, should be executed.

2. Scheduling Server

The Scheduling Server performs the highly complex task of scheduling multiple jobs, from multiple users, onto one (or several) computers from a pool of heterogeneous computing platforms. The sophisticated algorithms that the Scheduling Server will use to make decisions is beyond the scope of this thesis. However, it is essential to know the interfaces presented by the Scheduling Server.

The Scheduling Server will accept scheduling requests from the client libraries. The Scheduling Server will query both the Resource Status Server and the Resource Requirements Database. These queries will respond with near real time information on the status (load) of the VHM, and the resource requirements of the application. Once this information is obtained, the Scheduling Server can calculate a mixture of computing and network resources which will, with high probability, deliver the requested quality of service.

Additionally, in the event of a significant deviation from the initial resource status estimate, the Scheduling Server will receive notification from the Resource Status Server. For example, if a communications path is severed, or a machine fails, the Scheduling Server will be notified and can recalculate a new scheduling solution for the affected applications. The Scheduling Server may then signal the client library and advise it to compute using a different algorithm, or perhaps recommend that it shift execution to a different computing resource.

3. Resource Requirements Database

The Resource Requirements Database is a repository of information pertaining to the execution of user applications. A job consists of the code and data required to execute a user's application. The database records

statistics on the run time characteristics of jobs, such as CPU, memory, and disk usage. The Resource Requirements Database provides this information to the Scheduling Server as requested. It is updated by the client libraries.

4. Resource Status Server

The purpose of the Resource Status Server is to maintain a highly dynamic database of estimated loads on the various resources of the VHM [Ref. 6]. The Scheduling Server will query the Resource Status Server to obtain an initial estimate of the load on various compute resources. After making a scheduling decision, the Scheduling Server will notify the Resource Status Server of the additional loads that it expects the client application to place on the compute resources.

Also, during the execution of an application, the Resource Status Server will be updated periodically with statistics regarding the computing and networking resources in use by the client libraries.

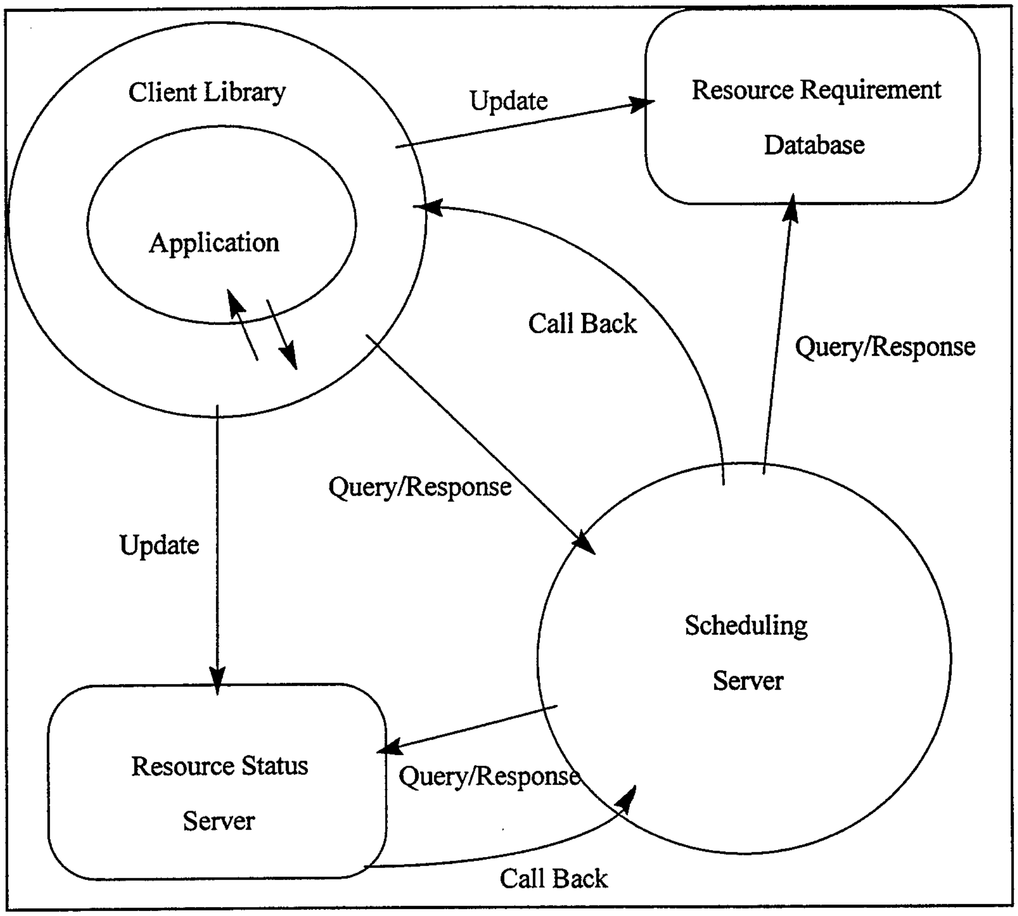


Figure 2 Overview of MSHN Runtime Architecture [Ref. 6]

C. SUMMARY

MSHN is a tool currently under development. Its purpose is to manage (adaptive) jobs in a heterogeneous environment, so that the system delivers good quality of service. Given the conceptual description from above, we shall now turn our attention towards the security aspects of MSHN.

IV. ESSENTIALS OF SECURE SYSTEMS

A. THE GOLDEN TRIANGLE OF INFORMATION SECURITY

Clearly, no computer system is 100% secure. There will always be some level of risk associated with the use of such a system. Therefore, it is more appropriate to describe the level of ***trust*** you have in that system, rather than calling it ***secure***. Trust, in this context, refers to the degree to which you believe the computing system will behave in the manner it was designed to behave. In other words, we expect the computing system to enforce a security policy. Information security is commonly viewed as a combination of the following features:

1. Secrecy:

Secrecy is defined as the protection of information from unauthorized disclosure.

2. Integrity:

Integrity is defined as the protection of information from modification or deletion.

3. Availability:

Availability is defined as the protection from denial of service attacks.

B. BUILDING A TRUSTED COMPUTER SYSTEM

The Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) [Ref. 7] describes the techniques that should be considered when developing a trusted computing system. The TCSEC has three control objectives:

1. Security Policy

A security policy is a statement of intent with regard to control over access to and dissemination of information. It must be precisely defined and implemented for each system that is used to process sensitive information. Two common policies are the *Mandatory Policy* and the *Discretionary Policy*.

A mandatory policy is one where the computer system enforces access control. One common implementation of mandatory policy uses a set of ordered labels that have been assigned to each data container and user. Access control decisions are determined by the relationship between the user's label and the container's label in the label hierarchy. Mandatory policies are often found in military applications, where users are assigned security clearances, and data are marked with a security classification. Only those users with a clearance equal to or higher than the data's classification label are permitted access.

Conversely, a discretionary policy allows the users of the computer system to decide who should receive access to the information they control, and the system enforces those access control decisions.

2. Accountability

The second basic control objective addresses one of the fundamental principles of security: ***individual accountability***. Individual accountability is the key to securing and controlling any system that processes information on behalf of individuals or groups of individuals. Accountability is supported by the following security mechanisms:

a) Identification

Each individual user of a system, whether they are an actual person or a process running on behalf of a user, must identify itself before obtaining access to the resources of the computer system.

b) Authentication

Authentication is the process of binding the identity of a user to who they say they are. Identification and authentication work together, and are often called I&A. A trusted computer system must have irrefutable proof that an identified user is legitimate, and not an impostor.

Once I&A has been established, then the trusted computer system can provide authorization control. In most computer systems, users may have differing access requirements depending on their role in the organization. Some users may require access to certain sensitive information that should not be revealed to other users. The administrator of such a system needs the ability to decide what level of access to bestow upon an authenticated user. Note that for this feature, granularity is an important attribute. Some applications may require access control per individual user, while others may wish to grant access to an entire group of users.

c) Audit

A trusted computer system should provide an audit capability. Thus, if an anomaly or a breach should occur, the audit trail could assist developers and administrators in their efforts to isolate and remedy the problem. Additionally, this feature can help investigators assess the extent of damage that may have occurred, and decide on an appropriate course of action.

Note that while auditing cannot prevent attacks, it does provide a psychological deterrence to insider crime. If a valid user knows that his actions are being recorded, he may be less likely to attempt an illegal action, even though the system might allow him to do so. Auditing is

extremely valuable as a tool for alerting appropriate personnel, recording the extent of a breach, and assisting in the eventual repair of security events.

3. Assurance

The third basic control objective is concerned with guaranteeing or providing confidence that the security policy has been implemented correctly and that protection relevant elements of the system do indeed accurately mediate and enforce the intent of that policy. By extension, assurance must include a guarantee that the trusted portion of the system works only as intended. To accomplish these objectives, two forms of assurance are required:

a) Life-Cycle Assurance

Life cycle assurance refers to the steps taken by an organization to insure that the system is designed, developed, and maintained using formalized and rigorous control and standards [Ref. 7].

b) Operational Assurance

Operational assurance focuses on the features and system architecture used to insure that the security policy is uncircumventably enforced during system operation [Ref. 7].

C. REFERENCE MONITOR CONCEPT

An abstract model of the operation of security mechanisms was proposed by James P. Anderson in 1972. His idea was called the Reference Monitor Concept [Ref. 8]. Figure 3 depicts the Reference Monitor Concept.

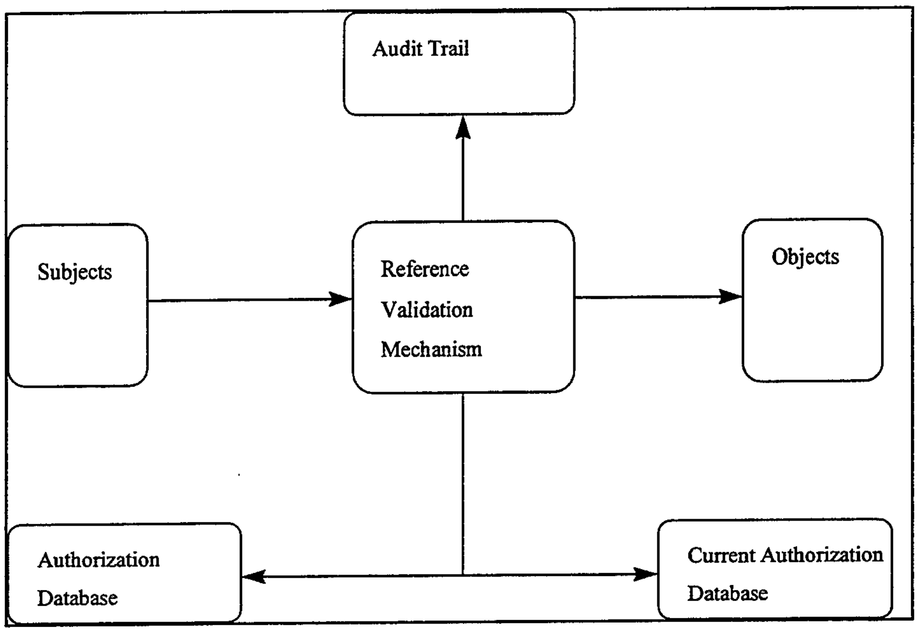


Figure 3 Reference Monitor Concept

The Reference Monitor Concept provides a theoretical framework for the design and implementation of security mechanisms which will enforce a particular security policy. The reference monitor allows active entities, called *Subjects*, to make reference to passive entities, called *Objects*, based on a set of current access authorizations. Subjects, such as users, and processes, cause information to flow among objects, or change the state of the system.

Objects, such as files, displays, and printers, are entities that contain or receive information.

The heart of the Reference Monitor is the Reference Validation Mechanism (RVM). The RVM arbitrates access requests. When the RVM receives a request for access from a subject, the RVM consults the authorization database. If the authorization database contains permission for the given subject/object pair, then access is granted, and the current authorization database is updated along with appropriate entries into the audit trail.

Any actual implementation of a reference monitor must satisfy three design requirements [Ref. 8]:

- Completeness: the reference monitor must be invoked upon every reference by a subject to an object,
- Isolation: the reference monitor and its database must be protected from unauthorized alteration, and
- Verifiability: the reference monitor must be relatively compact, organized, simple, and understandable so that it can be completely analyzed, tested and verified to perform its functions properly.

In practice, it is extremely difficult to produce an implementation of a reference monitor that fulfills these requirements. Software flaws will exist in all but the most carefully analyzed programs. Sophisticated software

engineering techniques and configuration control can certainly assist in the development of a high quality product, but they do not provide 100% assurance. Nonetheless, the reference monitor concept remains as an ideal, theoretical basis for the design of trusted systems.

D. SECURITY MECHANISMS

1. Identification and Authentication

Identification and authentication is a first step towards controlling access to information. It is a two-step process where the user first tells the system who they are, followed by proof to the system that who they say they are is in fact who they are. I&A mechanisms generally rely on one or more of the following three items:

- Something the user knows: the most common form of this technique is the use of a password,
- Something the user has: smart cards, tokens or identification badges are forms of something the user may have, and
- Something the user is: physical properties of the user's anatomy are examples of something he or she is. Finger prints, retinal scans, voice recognition and many other techniques are available to prove identity with a reasonable degree of assurance.

2. Cryptography

Cryptography is the science and art of secret writing - keeping information secret [Ref. 9]. Most people are aware of cryptography as a means for scrambling a message such that it can only be read by the holder of some secret code, however, confidentiality is only one of many services that cryptography can provide. Cryptography is an essential part of modern secure systems because of its ability to provide the following communications security services:

- Secrecy: protection against unauthorized disclosure,
- Authentication: undisputable proof of the originator of a message,
- Integrity: detection of unauthorized modification or deletion of a single bit of a message, and
- Nonrepudiation: undisputable proof of a transaction, such that neither party involved can deny the transaction after the fact.

Cryptography is implemented via encryption algorithms. Encryption Algorithms are mathematical functions that scramble data. There are numerous algorithms available. They can be divided into two categories: [Ref. 9]

- Symmetric Key Algorithms, and
- Asymmetric Key Algorithms.

a) *Symmetric Key technology*

Symmetric key ciphers use an algorithm for encryption and decryption with the same key (see Figure 4). Therefore, symmetric key schemes required that both the sender (User A) and receiver (User B) possess the same key in order to communicate in a secure manner. The confidentiality and authenticity of their data relies on the security of the key and the strength of the encryption algorithm employed.

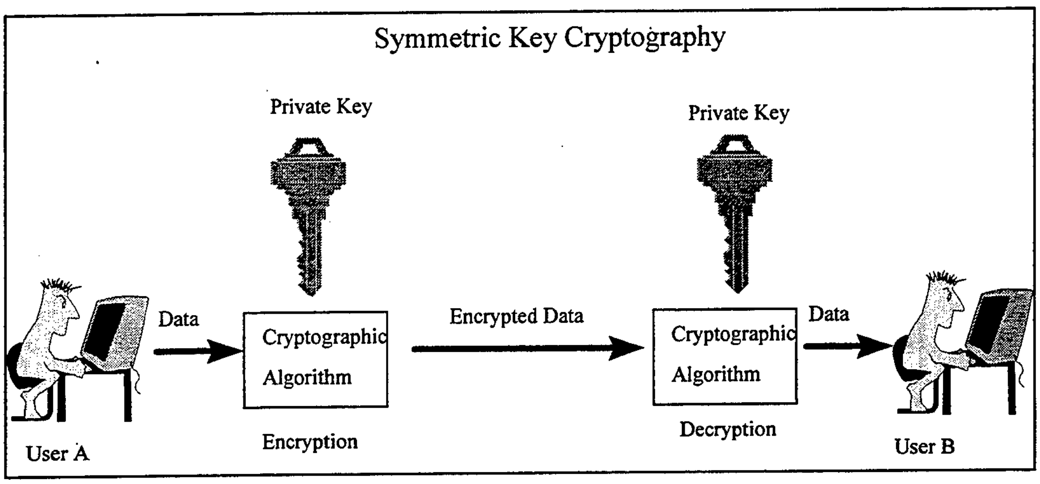


Figure 4 Symmetric Key Cryptography

Symmetric key systems require a trusted source for the generation and distribution of keys. The difficulty of symmetric key technology is the distribution process. Since the basis for trust lies in the secrecy of the keys, keys cannot be sent via unsecure means.

Key management in a symmetric system can be a non-trivial problem. Consider the fact that for each pair of users in an organization, there must exist a unique key. The total number of keys required can be calculated by the following formula:

N = [U * (U - 1)]/2

where N = Total number of unique keys, and U = number of unique users. Thus, if there are 100 users in an organization who wish to securely communicate encrypted messages, a total of [100 * (100-1)]/2 = 4950 keys would be required. Clearly, key management can get out of hand quickly, even in a small organization.

b) Asymmetric Key Technology

Cryptographic algorithms where one key is used to encrypt and a second key is used to decrypt are called asymmetric algorithms (see Figure 5).

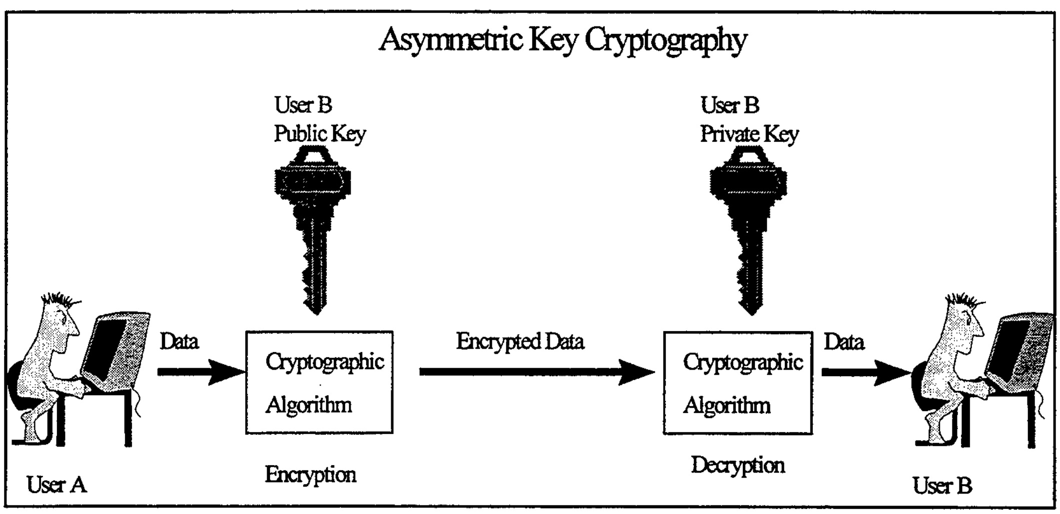


Figure 5 Asymmetric Key Cryptography

Asymmetric key systems offer significant advantages over symmetric systems when it comes to management and distribution of keys.

Public/Private key cryptography is a family of asymmetric key algorithms that offers several unique, and highly useful properties. The public/private key pairs (PPK) are generated such that when one key is used to encrypt, the other is used to decrypt. The user's public key is freely published for anyone to use, while the private key is kept secret. [Ref. 9]

Public key technology can be used to implement several important security mechanisms: confidentiality, authenticity, integrity and non-repudiation. To transmit a secret message from User A to User B, User A would encrypt the message using User B's public key. The only person who could decrypt the message would be the holder of User B's private key, which presumably, User B has kept securely to himself. User B can decrypt the message, but how does he know it came from User A? Anyone could create a message and encrypt it with User B's public key. This problem is solved by digital signatures, which are described later in this chapter.

Asymmetric key systems offer a significant advantage over symmetric systems in terms of key distribution. Since private keys remain secured with the originator, there is no need for a pre-planned secured

communications path or courier system to deliver keys. Public keys can be transmitted via electronic mail, posted on a web page, obtained via voice telephone call, or any other means, without regard to compromise.

The management of keys in an asymmetric system is much simpler than in a symmetric system due to the reduced number of keys required. For each user in the system, there are only two keys that need to be managed: the user's public and private key pair. Referring back to the previous example, in an organization of 100 symmetric key users, a total of 4950 keys were required. The same organization using asymmetric keys would require only 200 keys. This promises a significant workload reduction in terms of administrative costs. Public key systems are scalable. They easily expand with the growth of the enterprise.

Public key technology offers a wide range of advantages over symmetric techniques, however, it is not without its drawbacks. One significant disadvantage stems from the fact that the encryption and decryption algorithms are significantly slower. Consider the following comparison of two of the most popular cryptographic algorithms, Data Encryption Standard (DES), a symmetric algorithm, and the Rivest, Shamir and Adleman (RSA) Cryptosystem, a public key algorithm.

The most efficient current hardware implementations of RSA achieve encryption rates of about 600

Kilobits per second, as compared to 1 Gigabit per second for DES. Stated another way, RSA is roughly 1500 times slower than DES. [Ref. 10]

Even more important, the use of public keys requires an infrastructure designed to satisfy the following questions:

- How do I obtain someone's public key, and
- How do I know keys are valid?

Obtaining another user's public key may be as simple as placing a telephone call to the user and transmitting the key verbally. This simple solution might not always be practical. Another alternative is to post public keys on a web site that is open to anyone who might need your key.

The Consultative Committee for International Telegraphy and Telephony (CCITT) recommendation X.500 is a series of recommendations that define a directory service [Ref. 11]. The directory provides a means for mapping users to their public keys. Lightweight Directory Access Protocol (LDAP) is another example of an automated means for obtaining public keys.

Once the user has obtained the required public key, he or she must be assured that is legitimate, and not a forgery. Authentication of public keys is performed by electronically wrapping the key and other user data in a

certificate. The certificate is electronically signed by a trusted third party. This signature guarantees the authenticity of the certificate, much as a notary public performs the same function on paper documents.

c) *Digital Signatures*

A digital signature is a block of data that was created through the use of a cryptographic signing algorithm. The algorithm is applied to some input data using a private key. Digital signatures can be used for the following purposes:

- Authenticating the originator of a message or process, and
- Verifying the integrity of a message since it was signed by the originator.

Digital signatures provide authentication by virtue of the extreme computational difficulty of decrypting a signed message without the proper public key. The successful decryption of a digitally signed message with a public key virtually guarantees the message was signed by the holder of the corresponding private key.

Integrity of messages can be guaranteed by combining digital signatures with message digests. A message digest, or hash, is a small piece of data that is generated by processing the message through a hashing algorithm. A hashing algorithm uses a sophisticated

mathematical routine to produce a message digest that is extremely difficult to reverse. That is, given the hash, it is almost impossible to determine the message that was used to create it (there would be many).

Additionally, the hashing function is designed such that any change in the message will produce a completely different message digest. Therefore, if the originator of a message creates a message hash and signs it, the receiver of the message can verify the integrity of the message by recalculating the hash and comparing it with the one sent with the message. If they agree, then receiver can be reasonably certain that the message arrived without the change of a single bit.

Hashing algorithms and message digests suffer from a phenomenon known as *collisions*. The strength of a message digest is limited by the number of bits used to compose the hash. Normally, a message digest is small, perhaps several bytes, compared to the input message from which the hash was derived that may be several kilobytes to megabytes. Clearly, since there are more combinations for possible messages than there are for corresponding message digests, there will exist a number of messages that, when hashed, produce exactly the same message digest. This is a collision. The possibility of a collision implies that a person with sufficient computing resources and time can

generate a bogus message that produces the same hash as an authentic message, and therefore, break the integrity of your message. Thus, collisions and their risks should be considered when choosing a hash size and algorithm.

d) Digital Certificates

A digital certificate is an unforgeable binding between some user (person or process) to a public key in a particular domain as attested to by the digital signature of the owner of the certificate domain. Certificates can attest to the identity of the certificate holder and may include a set of authorized actions the holder may perform. CCITT recommendation X.509 defines a standard format for the content of certificates [Ref. 11].

The owner or creator of a certificate domain is known as the certificate authority (CA). The digital signature placed upon a certificate by the CA forms the basis for trust in a Public Key Infrastructure (PKI) system. The community of trusted certificates can be expanded by cross certification. When two or more CA's agree to trust each other's certificates, users can infer trust based upon this relationship (see Figure 6).

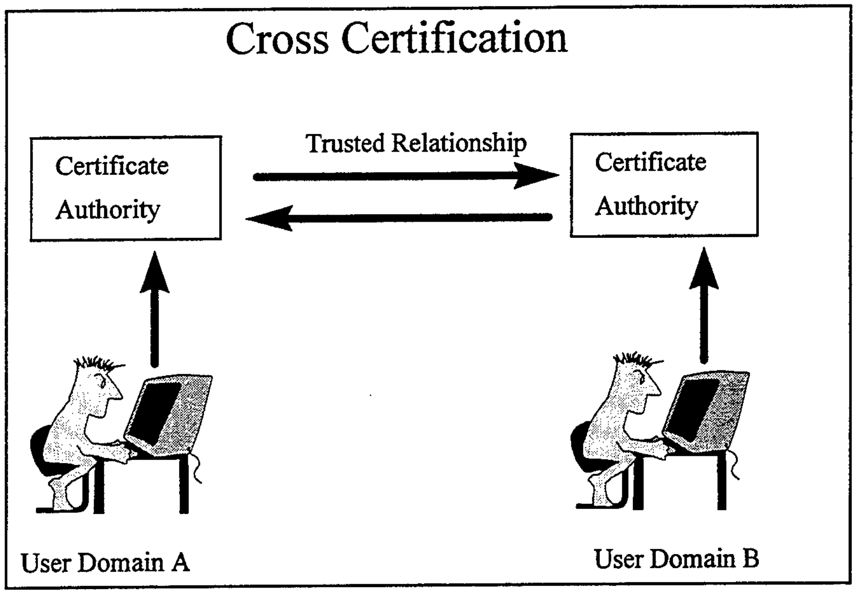


Figure 6 Cross Certification

Cross certification allows for the development of an expandable, hierarchical certificate authority structure. This infrastructure is essential to the success of public key technology. However, certificate authorities have another important job aside from the creation of certificates. What happens when a certificate is no longer valid? The revocation of certificates is a serious issue.

Certificate revocation can be handled in a number of ways. Usually, when a certificate is created it will include a field to indicate the expiration date of the certificate. The user of the certificate can check the expiration date prior to allowing any transaction, much the same way a credit card is checked for expiration before a purchase is committed.

There may be cases when a certificate authority wishes to revoke a certificate prior to its expiration date. For example, consider an employee who is terminated for cause. While the former employee's certificate is still within the validity date, the employee maintains the ability to identify himself as a member of the firm. This positive identity may grant him access to confidential data. Clearly, the firm would have the need to revoke the certificate immediately. Certificate Revocation Lists (CRLs) were developed for this purpose. A CRL, as the name implies, is a listing of certificates that are no longer valid. Now when a user wishes to check the validity of a certificate, they must check the digital signature of the signer as well as the revocation list. Provided that both tests pass, the user can be assured that the certificate is legitimate.

E. SUMMARY

Building a secure computing system is no trivial affair. A great deal of research has gone in to the development of sound security principles and mechanisms. Chapter V describes a commercial off-the-shelf product that implements security mechanisms.

V. COMMON DATA SECURITY ARCHITECTURE

In early 1995, Intel Corporation recognized the need and corresponding opportunity for a security infrastructure that was open, interoperable, and applicable across multiple computing platforms. The initial idea was to provide personal computers with a basic infrastructure so that the essential ingredients for a security solution were available. As a result of their research and development, Intel published a specification called the ***Common Data Security Architecture*** (CDSA) [Ref. 12].

Intel based their architecture upon a number of fundamental assumptions. These assumptions include design characteristics, and trust relationships.

In order to achieve their goal of interoperability and extensibility, the architecture was designed from the start to be built in a series of layers. Each layer is designed to provide services to the layer above it. The layered approach is modular, adaptable, and portable. Each of these features is highly desirable in any industrial-strength software engineering project.

Intel makes broad claims regarding the security of CDSA [Ref. 12]. Nonetheless, the security provided by CDSA is limited by the protection provided by the underlying operating system. Since cryptography forms the heart of

CDSA protection, the protection of keys and methods on each host becomes a critical aspect of security. If these keys are subject to unauthorized modification or disclosure, the application executing on top of CDSA is compromised. Thus, we see that unless carefully integrated into a high assurance platform, CDSA protection will only deter attackers with limited resources. The premise of our work is that CDSA may, in fact, be integrated into a platform having a level of assurance commensurate with the data.

Another architectural assumption that bears discussion is the representation of trust relationships. CDSA relies on digital certificates as the basis for identification and authentication functions. Certificates may also carry authorization information. CDSA does not mandate trust relationships or security policies, but it does allow applications to enforce them using its services.

CDSA defines the infrastructure for a comprehensive set of security services. The CDSA consists of three basic layers (see Figure 7):

- A set of system security services,
- The Common Security Services Manager (CSSM), and
- Add-in Security Modules.

The layered design of the architecture allows for extensibility of security mechanisms as future enhancements become available. In addition to the vertical layers, the security modules are horizontally divided by function. Each of these layers is described in detail in the following paragraphs.

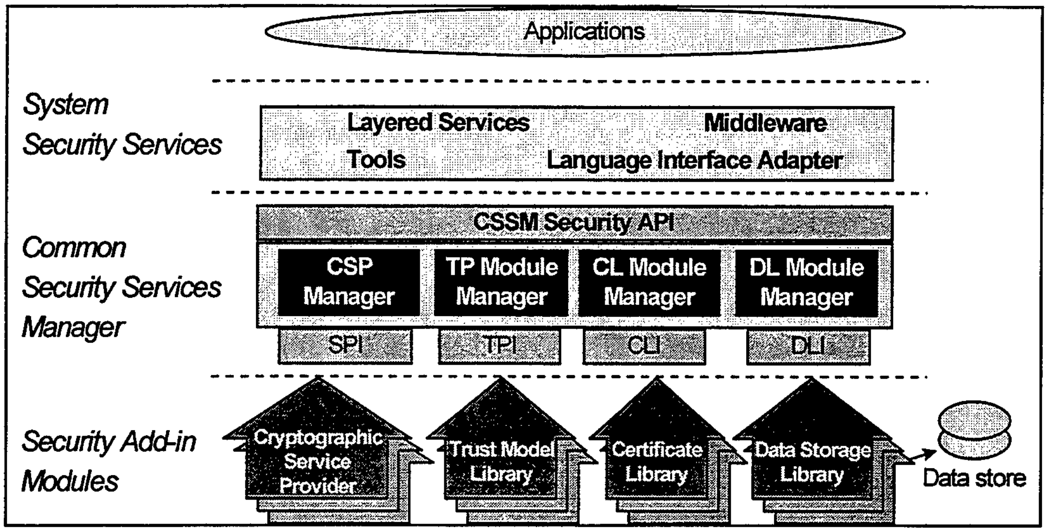


Figure 7 Common Data Security Architecture [Ref. 12]

A. SYSTEM SECURITY SERVICES

Sophisticated security protocols, based on the Common Security Services Manager and its add-in modules, may be defined and implemented at the system security services architectural layer. CDSA documents claim that the following services can be provided: [Ref. 12]

- Secure and private file systems,
- Protocols for secure electronic commerce,

- Protocols for private communications,
- Multi-language access to the CSSM API, and
- CSSM management tools.

Examples of these protocols may include Pretty Good Privacy (PGP) [Ref. 13], Secure Electronic Transaction (SET) [Ref. 14], Secure Sockets Layer (SSL) [Ref. 15], and Secure Multipurpose Internet Mail Extensions (S/MIME) [Ref. 16]. These protocols support secure file systems, secure electronic commerce, secure network communications, and secure electronic mail, respectively. An example of multi-language support is the CSSM-Java API, which allows Java developers the capability to include CSSM functions in their applications. CSSM management tools may include installation and configuration utilities.

B. COMMON SECURITY SERVICES MANAGER (CSSM)

The main infrastructure component of CDSA is the Common Security Services Manager. The CSSM integrates the security functions required by applications programs. From the application developer's point of view, this is a great idea, because it facilitates the design and implementation of the final product. It allows the developer to concentrate on his primary mission, i.e., producing an application program, rather than the low level details of complex cryptographic algorithms. The CSSM services are organized into the following categories:

■ CSSM Security API,

■ CSSM Core Services, and

■ Module Managers.

The services provided are shown below in Figure 8.

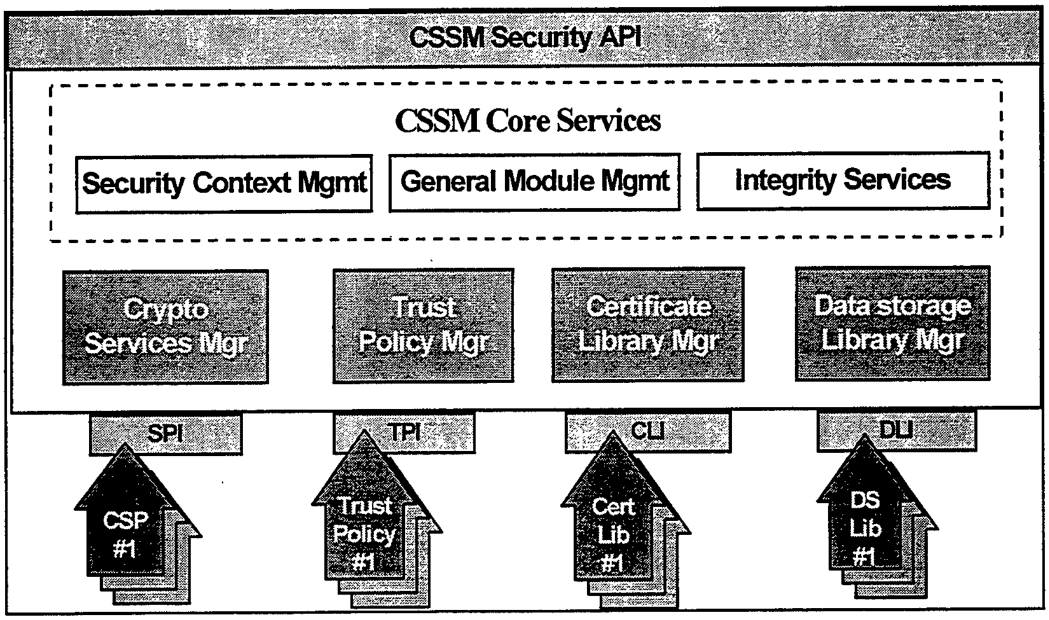


Figure 8 CSSM Services [Ref. 12]

1. CSSM Security API

The CSSM Security API is the defined interface through which application programs access security services. The API includes hundreds of function definitions related to security. Applications may request security services directly through the CSSM Security API or indirectly via layered security services and tools implemented over the CSSM API. In addition, the API provides a means for

accessing service-provider, module-specific mechanisms via *pass-through* functions.

2. CSSM Core Services

Services that pertain to the installation and management of the CSSM are *Core Services*. These services are implemented by the CSSM, not by add-in modules. The four Core Service categories are described below.

a) Security Context Management

A security context is a run-time data structure that contains the security related parameters required to execute a cryptographic function. In order to perform cryptography, applications programs must gather numerous attributes, such as algorithm identifiers and key sizes. These attributes are aggregated by a call to a security context creation function. A handle to the new security context is returned to the application. The application may use the context handle as an input to a cryptographic function along with the data to be operated upon.

Security context management functions handle the creation and deletion of contexts. The use of security contexts is beneficial because of their reusability. Once the overhead of creating a context has been spent, there is no further performance penalty for reusing the context as needed. An application that repeatedly performs a CSSM API

function can improve its execution performance by saving context handles for future function calls.

b) General Module Management

An *add-in module* is a product provided by a hardware or software vendor that performs security services. In order for an application to use the functions of an add-in module, the add-in module must be selected and installed (attached). Information pertaining to attached components is stored in a registry. The module management functions of the CSSM allow an application to query the features and status of add-ins. The query may return descriptive attributes specific to the particular add-in. The application can use the results of the query to selectively determine which add-in modules to dynamically load and use. This feature could be useful in a dynamic operational environment where the security requirements of the application are subject to change. Naturally, the module management also offers functions for detaching and uninstalling modules after they are no longer required.

c) Integrity Services

Protection of CSSM components from tampering is provided by *integrity services*. Upon creation, the CSSM itself is digitally signed. At the time of installation, the CSSM will verify its signature. This assures the user

that the CSSM is authentic and has not been modified since it was distributed from the manufacturer.

Additionally, the integrity of add-in modules can be checked as they are dynamically loaded. The CSSM performs an authentication procedure as part of the add-in module attachment process. If the authentication fails, the attachment will abort, and the add-in module will not be available to the application program.

CDSA's notion of self-protecting integrity domains is flawed, without an underlying domain mechanism. If a malicious element is able to control the portions of the system upon which CDSA depends, it can completely subvert CDSA integrity. Once again, we see that CDSA is only as trustworthy as the underlying computing platform.

d) Memory Management

CSSM-created objects require allocation of random access memory for non-persistent storage. Therefore, the CSSM includes functions for the management of memory resources. Application developers have the option of allocating memory within the application program, or requesting the CSSM to allocate it from the CSSM's own memory heap. This memory management option supports the broadest range of potential architectures. Applications can use the CSSM memory management functions to free memory dedicated to objects that are no longer needed.

3. Module Managers

The matching of a CSSM Security API function call to the appropriate Service Provider Interface (SPI) function is the job of the module manager. The CSSM has organized security services into four basic categories and has defined a module manager for each service as follows:

- Cryptographic Services Manager,
- Trust Policy Services Manager,
- Certificate Services Manager, and
- Data Store Services Manager.

Each module manager administers the corresponding add-in module installed on the local system. Module managers define the API for their particular module.

C. SECURITY ADD-IN MODULES

Cryptographic operations, security policy decisions, and certificate manipulation operations are performed by security add-in modules. Application developers may purchase add-in modules from hardware and software vendors, thus freeing themselves from the burden of developing security specific code. Add-in modules are divided into four functional subsets:

1. Cryptographic Service Provider Module

The CSSM does not perform cryptographic algorithms. Rather, the CSSM provides applications programs with access

to cryptographic mechanisms implemented via Cryptographic Service Provider (CSP) modules. For this reason, the CSSM is known as *crypto with a hole*, where the hole is filled by the CSP vendor's product [Ref. 12].

The benefit of modular CSPs lies in their exchangeability. Application developers can pick and choose their CSP as required. CSPs may be implemented in hardware or software. The application requesting CSP service uses the same API function calls in either case. However, we would expect that a hardware implementation would be more tamper resistant than a software CSP.

A general purpose CSP can be used to perform the following cryptographic functions and services: [Ref. 12]

- Bulk Encryption and Decryption,
- Digital Signing and Verification,
- Cryptographic Hash,
- Key generation,
- Random Number Generator, and
- Encrypted Storage of private keys.

Specific instances of CSPs may include more or less functionality as determined by the individual service provider.

2. Trust Policy Module

The primary purpose of the trust policy module is to answer the question "Is this certificate trusted for this action?" [Ref. 12] Thus, application developers can place the policy-specific business rules of their application into one module. Whenever an access decision needs to be made, the application forwards the requester's certificate, and requested action to the trust policy module for review. The trust policy module will respond appropriately, based upon the access control rules programmed into the module.

3. Certificate Library Module

The certificate library module performs operations on memory-resident certificates and certificate revocation lists (CRLs). CDSA defines the following operations that any certificate library module should be able to perform: [Ref. 12]

- Create new certificates and new CRLs,
- Sign existing certificates and existing CRLs,
- Verify the signature of existing certificates and CRLs,
- Extract values, such as a public key, from certificates,
- Import and export certificates of other data formats,

- Revoke certificates,
- Reinstate revoked certificates, and
- Search certificate revocation lists.

CDSA does not mandate any particular certificate format. Those design decisions are left up to the module developer. Therefore, as new standard certificate formats are designed and accepted by industry, updated certificate modules can be employed without have to redesign the application program.

4. Data Storage Library Module

The primary purpose of the data storage module is to provide secure, persistent storage for certificates, and certificate revocation lists [Ref. 12]. A data storage library module performs the following operations:

- Opening and closing data stores,
- Creating and deleting data stores,
- Importing and exporting data stores, and
- Data object manipulations, such as insertion, update, deletion, and retrieval.

Service providers may design and implement data storage library modules from scratch or they may take advantage of an underlying commercial database management system (DBMS). Either way, the application developer need not be concerned with the details of storing data objects.

D. SUMMARY

Since its release for public review, Intel's CDSA has generated support from the computing industry. In January, 1998, The Open Group announced that it had adopted Intel's Common Data Security Architecture (CDSA 2.0) specification as an industry-accepted specification for the development of secure applications that are interoperable, extensible, and offer cross platform support [Ref. 17]. MSHN can benefit from the services provided by CDSA. Chapter VI describes a security-enhanced MSHN architecture, along with a prototype demonstration that uses CDSA security mechanisms.

VI. SECURITY ENHANCED MSHN ARCHITECTURE

Given the initial MHSN architecture described in Chapter III and the requirements for security as discussed in Chapter IV, we now describe an enhanced MSHN architecture and prototype demonstration, designed to satisfy a limited security policy.

A. ASSUMPTIONS

1. Application Level Security

MSHN is designed to execute in a heterogeneous computing platform environment. Machines within the VHM may have differing operating systems, central processing units, and communications protocols. MSHN acts as middleware: it will not require any OS modifications, nor will it rely upon any of the unique security features of any particular OS or CPU.

MSHN executes in user mode. Each of MSHN's components runs with the privileges of the user who invoked it. It is intended that MSHN not have arbitrary control over the host executing it. This implies that MSHN should not have supervisory, i.e., “root”, privilege within a system. Therefore, MSHN can provide confidence of correct security policy enforcement only to the extent that any application can. The dependency of MSHN upon the operating system underlying it determines the assurance environment. User-

level middleware will only be as secure as the operating system and hardware it relies upon. No matter how well any user-level middleware is able to deny an attack, it will still be susceptible to penetration if the underlying OS is weak. The dependency upon underlying mechanisms is also true of other resource management systems, such as Globus [Ref. 18], and Legion [Ref. 19] which exist above the operating system. They are vulnerable to attacks via underlying layers.

2. User Identification and Authentication

Each user will have an individual account on every machine they are authorized to use throughout the VHM. Therefore, the assumption is that the I & A facilities (i.e., login and password) of the user's client machine along with those of the compute resources will be used to verify the identity and authenticity of the user. (Details of the overall MSHN architecture are still under development and the use of remote login facilities may be considered). For each user, MSHN will maintain a list of the machines where that user has an account. The Scheduling Server will consult this list as part of its algorithm for determining which machine to run a job on.

3. Public Key Infrastructure

In this thesis, we suggest that MSHN rely on public key technology for the implementation of security mechanisms.

Our enhanced architecture assumes the existence and availability of standard directory services such as Lightweight Directory Access Protocol (LDAP) or X.500. Additionally, we assume that some form of standard certificate, such as X.509 will be available.

This approach to key management is in contrast with that taken by the National Computational Science Alliance, of which Globus is a member [Ref. 1]. Their approach uses Pretty Good Privacy (PGP) public key encryption software for securing messages. PGP relies upon *introductions* for establishing trust relationships. These relationships are fundamentally ad hoc, whereas X.500 is authority based, and therefore more appropriate for the Department of Defense [Ref. 20].

The Legion resource management system eschews any notion of hierarchical trust. For this system, it is proposed to tie key management and authentication directly to each object with the support of an authorizing component in the architecture [Ref. 19]. Thus, each object proclaims itself, and other entities in the system must interpret that proclamation. See [Ref. 21] for further discussion of related work.

It is assumed that the MSHN Core components (Scheduler, RSS, RRD) each have a public/private key pair available. To simplify the architecture, the Scheduler, RSS, and RRD will

share a common MSHN core certificate. Hence, the MSHN core components will also share a common public/private key pair. Additionally, each user can obtain a certificate that provides a binding between the user's identity and his/her public key.

4. Compute Resources

MSHN will maintain a database of registered compute resources (including network and file and database servers). MSHN core components will be responsible for adding, deleting and updating this database. All of the code and data required to run the job will either be packaged by the client library and transported to the compute resource for execution, or the MSHN execution shell at the compute resource will fetch the code and data from the appropriate locations. Since our revised architecture will rely upon the Public Key Infrastructure, there must exist facilities for the storage of private keys at each compute resource. The prototype demonstration will use the key storage facilities provided by the CDSA crypto-service module to satisfy this requirement.

5. Client Library

It is assumed that the client library is benign, that is, free from malicious code.

6. User Intent

It is assumed that since the users do not have to use MSHN to run their jobs (they already have permission to execute jobs on machines in the VHM), they will not intentionally mislead MSHN with bogus run-time parameters.

B. MSHN SECURITY POLICY

Based upon the assumptions above and their implied restrictions, we now describe the security policy for our proposed enhanced MSHN architecture.

- MSHN will use cryptographic techniques to create separate domains between the MSHN core components and the user's application,
- MSHN will protect its databases (RSS/RRD) from malicious updates attempted by non-MSHN jobs,
- MSHN will authenticate messages communicated between its core components such as Query/Response messages between the Scheduling Server and the RSS/RRD,
- MSHN will provide guaranteed integrity of communications, if required by the user,
- MSHN will provide confidentiality protection of communications (encryption) if required by the user, and
- MSHN will provide an audit trail.

C. DOMAIN FRAMEWORK

The notion of a program integrity policy was proposed by L. Shirley and R. Schell [Ref. 22]. Program integrity means that modification to executable programs by untrustworthy subjects is prohibited. In essence, it is a policy to ensure that more sensitive programs remain tamperproof. Shirley and Schell recommend solving the program integrity problem by assigning programs to an ordered set of access classes. The access classes map to separate domains, and an underlying kernel mechanism enforces the integrity policy.

The basis for protection in the enhanced MSHN architecture is the separation of components into domains via cryptographic techniques. Figure 9 displays the MSHN domain framework. The domains are ordered in terms of privilege. From MSHN's point of view, user applications are the least privileged and are granted the least trust, while MSHN core components are more privileged and are granted a higher level of trust.

Additionally, users are separated from one another by relying upon authentication and a unique session key for each job submitted to MSHN. The use of a per job session key also allows the MSHN core components to guarantee the authenticity of status messages sent by executing user jobs and to reject bogus database updates.

By creating domains, the damage incurred by a security incident, such as the compromise of a key, is limited to the components of that domain.

The intent of the enhanced security architecture is to develop a framework that can take advantage of program integrity mechanisms, such as those described by Shirley and Schell, that would be available in a high assurance platform.

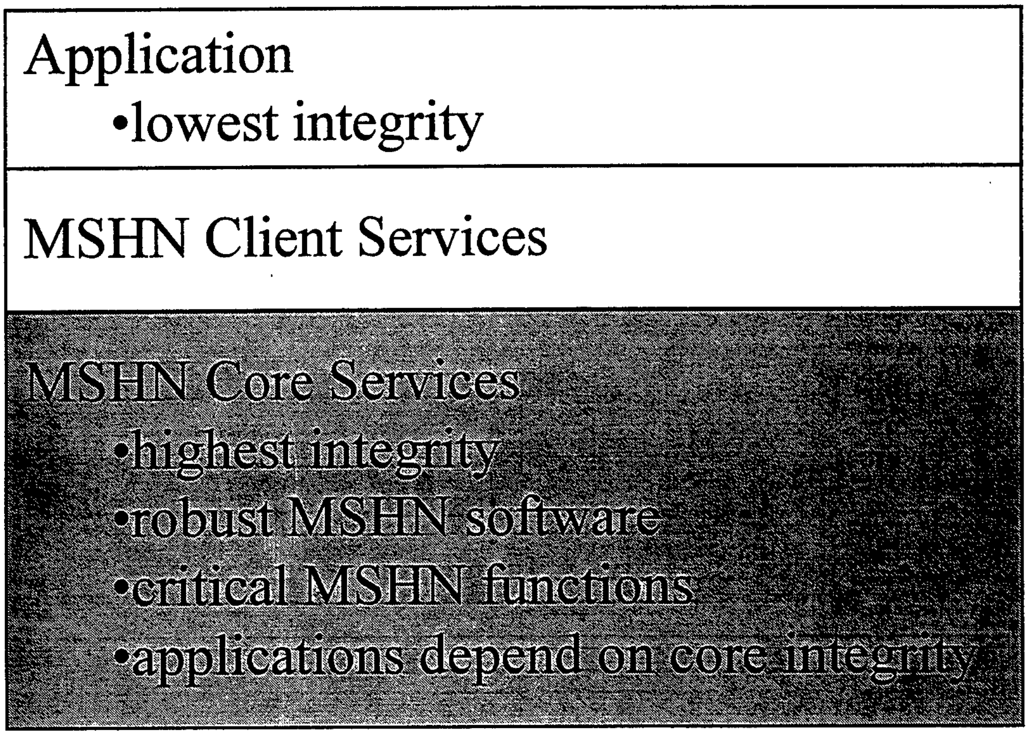


Figure 9 MSHN Domain Framework

The computers that MSHN will execute upon do not necessarily have the ability to run applications in separate domains. Through the use of cryptography, this domain

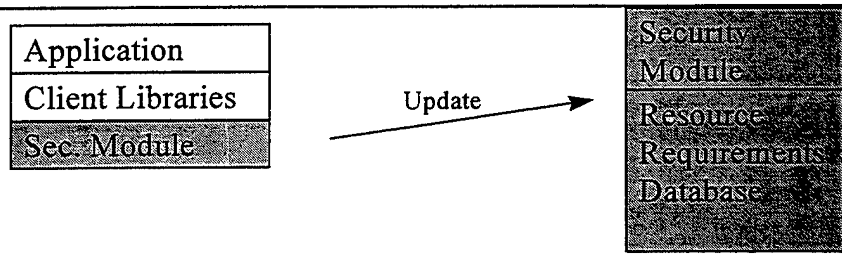
framework will allow all MSHN resources to maintain separate lightweight security domains.

D. MSHN SECURITY MECHANISMS

The enhanced MSHN architecture will provide the following security mechanisms:

- Administrative Control,
- Authenticated RSS/RRD updates,
- Authenticated Query/Response and Call Back Signals,
- Audit Trail, and
- Protected Communications (Encryption).

Authentication of messages between MSHN components using public key technology is relatively straight forward. Assume each component has a private key maintained in a secure manner. Also assume that each component has a corresponding public key which is published in a known directory service. Then, each component could be assured of authentic messages using the protocol described in Figure 10.

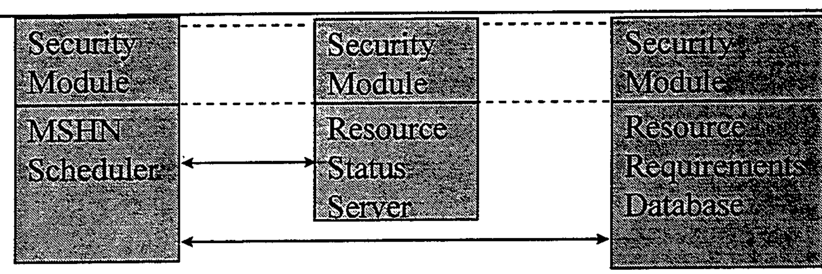


1. Client Library generates an Update message, and passes it to the CL Security Module.
2. Security Module digitally signs the Update message using its private key.
3. Security Module transmits signed Update message to RRD.
4. Security Module optionally creates an audit record and sends it to the audit server.
5. RRD receives the signed Update message and calls its Security Module to verify the signature using the Client Library's public key.
6. If the signature verification is successful, the RRD is assured that the Update message could only have come from an authentic MSHN Client Library.

Figure 10 PKI Authentication Protocol

We note that this protocol does not ensure the confidentiality of the Update message. If the communications path between the components is subject to eavesdropping, then the unauthorized disclosure of the Update message is possible. If confidentiality were required, then the Update message would require an additional step before transmission. First, the Update would be digitally signed with the originator's private key (guaranteeing authenticity). Second, the signed Update message would be encrypted, using the receiver's public key (guaranteeing privacy). This same protocol could be used in a similar manner for the authentication and confidentiality of messages between any of the components of MSHN.

An alternative, and more efficient method for the authentication of messages between MSHN components would be to use public key techniques to distribute a symmetric communications session key to all the components. Once the initial overhead of distributing the symmetric key has taken place, then encryption of messages could be performed using the much faster symmetric algorithms. This method would protect the authenticity of the message and guarantee privacy. Figure 11 describes the hybrid public key distribution, symmetric bulk encryption method.



1. MSHN Scheduler Security Module generates a symmetric session key along with a message digest of the same key. The digest is encrypted with the MSHN Core private key. The key and digest are bundled and encrypted with the MSHN Core public key, and transported to the other MSHN Core components.

2. Security Module at RSS and RRD decrypt the bundle using their MSHN Core private key. Privacy of the bundle is guaranteed because only the holder of the MSHN Core private key could decrypt it.

3. Security Module at RSS and RDD decrypt the message digest of the session key using their MSHN Core public key. Authenticity of the digest originator is guaranteed because only the holder of the MSHN Core private key could have encrypted the digest.

4. Security Module at RSS and RDD generate a message digest of the session key and compare it with the digest included in the bundle. If the digests match, the integrity of the session key is guaranteed.

Figure 11 MSHN Core Component Session Key Distribution

The communications protocols described earlier have not been formally evaluated. The actual implementation of the key exchange and message authentication procedures may require the use of a timestamp and/or nonce (one time use number or identifier) [Ref. 13] to prevent replay attacks.

E. REVISED ARCHITECTURE

The implementation of the MSHN security mechanisms will require that several components be added to the architecture described in Chapter III. A high level view of the revised architecture is shown in Figure 12.

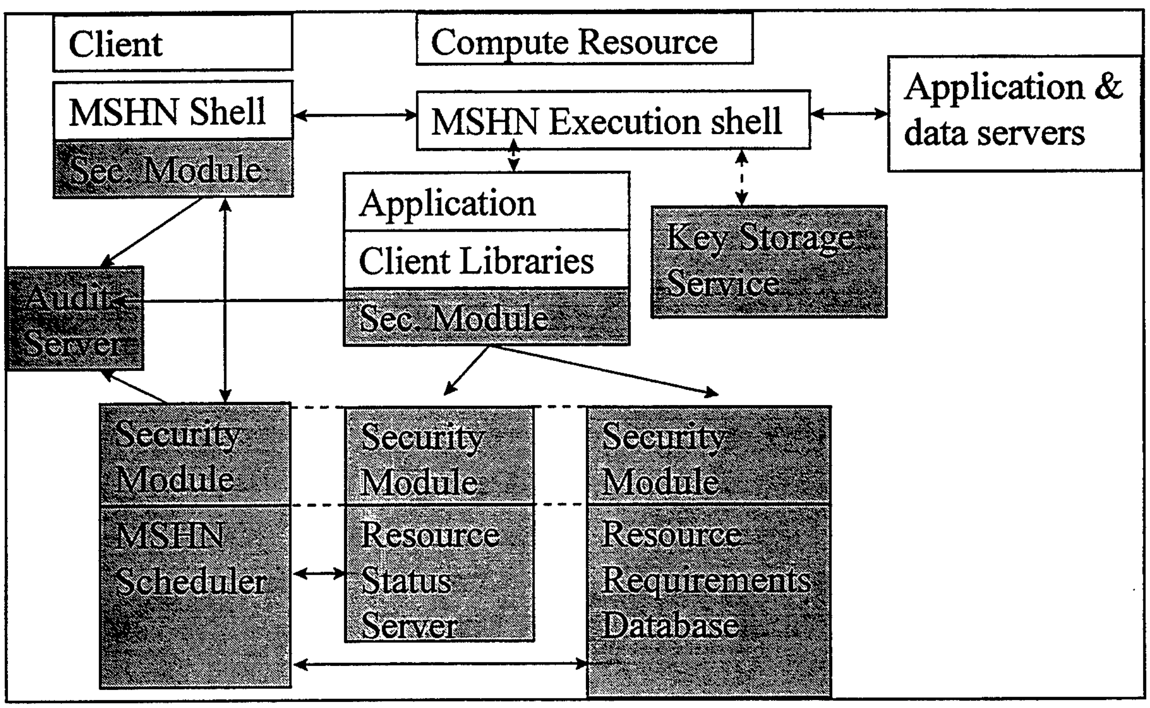


Figure 12 MSHN Revised Architecture

1. Security Modules

Each of the original components: Scheduling Server, Resource Status Server, and Resource Requirements Database will be augmented by a Security Module. The purpose of the Security Module is to perform the following cryptographic operations:

- Generate symmetric keys and asymmetric key pairs,
- Calculate Message Hash (Digest),
- Encrypt/Decrypt (both symmetric and asymmetric algorithms),
- Verify Signatures, and
- Sign Messages.

The Security Module at the Scheduling Server has the additional responsibility of creating and maintaining a MSHN Job Request Database.

2. Key Storage Service

A secure key storage facility will reside and execute at each compute resource within the VHM. Each compute resource must have a unique PPK pair. The private key of that pair will be stored and protected by the key storage service. The private key will be used to guarantee that the job sent to the compute resource is only readable by the bona fide resource.

3. Audit Server

The audit server will be the repository for the audit log. As transactions are processed throughout the MSHN components, applicable audit entries will be forwarded to the audit server for inclusion into the log. The audit server will provide an interface through which the MSHN administrator can review previous MSHN activity, and modify the list of events to be audited.

F. MSHN EXECUTION SCENARIO

The best way to describe the function of each of the MSHN components is to walk through a sample job execution flow from initiation by the user to return of the output from the compute resource. The following notation will be used to describe the cryptographic procedures:

- **E**key[]: Encrypt contents of [] using key
- **D**key[]: Decrypt contents of [] using key
- **S**key(): Create a Digital Signature of () using key
- K_s: Symmetric MSHN core session key
- K_j: Symmetric MSHN job session key
- U_{pri}: User's private key
- U_{pub}: User's public key

- U_{cert}: User's certificate
- $Core_{pri}$: MSHN core private key
- $Core_{pub}$: MSHN core public key
- $Core_{cert}$: MSHN core certificate
- R_{pri}: Compute resource private key
- R_{pub}: Compute resource public key
- Dig: Message Digest

The following discussion describes a typical job execution scenario, assuming that the user choose both the integrity and confidentiality security option.

1. Initialization

Prior to the execution of any jobs, a MSHN initialization procedure takes place. During this procedure, the MSHN core components, i.e., Scheduler, RSS, and RRD, will synchronize on a unique symmetric session key. The symmetric key will be used for bulk encryption due to its significant performance advantages. The initialization procedure is invoked by the Security Module on the Scheduler (or other designated MSHN core master server) at the request of the MSHN Administrator. The MSHN Administrator will start the Scheduler and supply it with a parameter specifying the required MSHN Core communications security option. The MSHN Core communications security option ranges from no security checks, to integrity check,

confidentiality, to a combination of integrity and confidentiality.

The Security Module on the Scheduler will generate a unique, symmetric MSHN session key (K_s). The security module will generate a message digest (Dig) of the MSHN session key and encrypt the message digest with the MSHN core private key. Then it will bundle the MSHN Session key with the encrypted digest and encrypt the bundle with the MSHN core public key.

$$\text{Bundle} = \mathbf{E}\text{Core}_{pub}[K_s, \mathbf{E}\text{Core}_{pri}(\text{Dig}(K_s))]$$

The bundle will then be transmitted to the other core components.

The Security Module on the RSS and RRD will receive the bundle from the Scheduler and decrypt it using the MSHN core private key. The message digest will be decrypted using the MSHN core public key. Next the MSHN session key will be verified by generating a message digest of the key and comparing it to the digest that was included in the bundle. If the verification test passes then we have established the secure transmission of a session key at each of the MSHN core components and MSHN is prepared to accept user jobs.

2. MSHN Job Request

The user at the client machine starts the MSHN shell. The user will specify the required communications security option. All the job information, parameters and the user's

certificate are bundled and, if requested, signed with the user's private key, and encrypted with the MSHN Core public key.

$$\text{Bundle} = \mathbf{E}Core_{pub}[\mathbf{S}U_{pri}(U_{cert}, \text{Job Info})]$$

The bundle is then transmitted to the Scheduling Server.

3. Scheduling

The Security Module at the Scheduling Server will receive the user's job request bundle and decrypt the bundle using the MSHN Core private key. The Security Module will verify the user's certificate based upon the certificate validation level and verify the signature on the inputs using the user's public key from the enclosed certificate. If the verification test passes, the Security Module will generate a unique request identifier, followed by the creation of an entry in the Job Request Database containing the user's identification, security option required, and request identifier.

The Scheduler will now gather the information it needs to calculate a scheduling solution. It will query the RRD and RSS. These queries and responses will be encrypted and or authenticated as required by the MSHN core communications security option parameter. These communications will be encrypted and/or signed with the MSHN session key (K_s) that was distributed at initialization time.

The Scheduler will calculate a solution based upon the data returned from the RSS, RRD, and the resources accessible to the user. The Security Module on the Scheduler will generate a unique job session key (K_j). The job session key will be added to the job request database, and the job request will be distributed to the RSS and RRD security modules.

The Scheduler security module will create a security token, which contains the job session key encrypted by the compute resource's public key. The job session key will be encrypted with the user's public key. All items will be signed using the MSHN core private key, and all these items, along with the Scheduler's certificate, will be encrypted by the user's public key and bundled as follows:

$$\text{Bundle} = \mathbf{E}U_{pub}[\mathbf{S}Core_{pri}(\text{Job info}, \mathbf{E}u_{pub}[K_j], \text{token}(\mathbf{E}R_{pub}[K_j]), Core_{cert})]$$

This bundle will be transmitted to the MSHN shell on the client machine.

4. Application Execution

After receiving the job information from the Scheduler, the MSHN shell will decrypt the bundle using the user's private key and verify the signature of the job data using the Scheduler's public key. It will decrypt the job session key using the user's private key. The client now knows which compute resource to use. The client bundles the

user's certificate, job info, and security token. The bundle is signed and encrypted.

$$\text{Bundle} = \mathbf{E}R_{pub}[\mathbf{S}U_{pri}(U_{cert}, \text{token, job info})]$$

The bundle is then transmitted to the appropriate compute resource.

The compute resource accepts the bundle from the client and decrypts it and verifies the signature. The compute resource decrypts the job session key that was extracted from the security token of the bundle. The compute resource executes the job application on behalf of the user and collects the output. The application output is signed and encrypted before being transmitted back to the client.

Once the application has completed, and possibly during execution of the application, the compute resource will forward job statistics to the resource requirements database and the resource status server. These status messages are encrypted and signed with the job session key before transmission to the respective database server.

The RSS and RRD accept incoming job statistic messages. The security module at each server will retrieve the appropriate job session key from the job request database, as specified by the request identifier. The messages are decrypted and verified using the job session key before they are entered into the database.

The client accepts the application output from the compute resource and decrypts/verifies the output before presenting it to the user. At this point, the client has finished, and will wait for the user to submit another job or terminate the program.

G. PROTOTYPE IMPLEMENTATION ARCHITECTURE

A complete implementation of the MSHN architecture is beyond the scope of any one thesis. However, we have developed a prototype security implementation to demonstrate how a MSHN server might communicate securely with a MSHN client using CDSA.

1. MSHN Security Layer

The foundation of the prototype implementation is encapsulated in a single C++ class definition. This class is called the MSHN Security Layer (MSHN_SL). The purpose of the MSHN_SL is to provide an interface through which MSHN can access the underlying security mechanisms provided via CDSA. C++ was chosen as the prototype language because Intel's reference CDSA implementation is written to support C++. We choose to execute our prototype in a Windows NT PC environment because this was the operating system upon which Intel's CDSA was implemented.

We created this layer as an interface between MSHN and CDSA so that, in the event that a competitive security API

is chosen to implement MSHN security, rather than CDSA, MSHN will still function with minimal reprogramming[2].

2. MSHN SECURITY SERVICES

The MSHN Security Layer defines the methods that are invoked to support the MSHN security policy. These security services are defined below, and a complete listing of the source code is included in Appendix A.

a) mshn_sl_init:

This method performs initialization tasks as required by the underlying security API.

b) mshn_sl_create_cert:

This method will create a certificate to include the public-private key pair associated with it. The certificate created will contain the public key, while the private key is placed into a secure key storage facility.

c) mshn_sl_get_cert:

This method will search the certificate database for a certificate whose subject name matches the input parameter and if found, returns the requested certificate.

---

[2] Common Object Request Broker Architecture (CORBA) [Ref. 23] and its security services [Ref. 24] have been under consideration since December, 1997.

d) mshn_sl_cert_verify:

This method will check the signature on a given certificate and return a Boolean value based upon the signature verification.

e) mshn_sl_cert_revoked:

This method will check a given certificate against a certificate revocation list and return a Boolean value depending on the revocation status.

f) mshn_sl_get_public_key:

This method will return a public key obtained from a given certificate.

g) mshn_sl_get_private_key:

This method will return a reference to a private key given the associated public key.

h) mshn_sl_put_audit:

This method will post a transaction to the MSHN audit server.

i) mshn_sl_encrypt:

This method will accept a data buffer and key, and return an encrypted copy of the input data.

j) mshn_sl_decrypt:

This method will accept a data buffer and key, and return a decrypted copy of the input data.

k) mshn_sl_sym_key_gen:

This method generates a symmetric key.

l) mshn_sl_asym_key_gen:

This method generates an asymmetric (public/private) key pair.

m) mshn_sl_digest:

This method creates a message digest for a given input buffer of data.

n) mshn_sl_sign:

This method creates a digital signature for a given buffer of data.

o) mshn_sl_sig_verify

This method accepts a signature and data buffer. It returns a Boolean value depending on the verification of the signature against the input data.

H. PROTOTYPE DEMONSTRATION

Once the MSHN_SL class was designed and tested, the next step was to develop sample programs to simulate the MSHN Client, MSHN Servers: Scheduler, Resource Status

Server, Resource Requirements Database, and the Compute Resource. We created five programs to test our prototype. The programs executed on three separate personal computers connected on a local area network (LAN). One PC served as the client. Another PC served as the compute resource, and one PC served as the MSHN scheduler, MSHN RSS and MSHN RRD, each of which executes as a separate process. These programs are described below and a complete listing of the source code is included in Appendix B.

1. Client

This program simulated the user interface to MSHN. The client accepts requests for MSHN jobs from the user and initiates the MSHN Scheduler. After receiving scheduling advice, the Client submits the job to the Compute Resource. Once execution of the job is complete, the Client receives the output from the Compute Resource, and after verification and/or decryption, returns it to the user.

2. Resource

Resource substitutes for the actions taken by the Compute Resource. Upon start-up, Resource goes into a wait state. It monitors the LAN until it receives a MSHN job to execute. Upon receipt, it will decrypt and/or verify the job request bundle and execute the job. The job output is transmitted to the Client, and job statistics are forwarded to the MSHN servers (RSS/RRD).

3. MSHN Core: Scheduler, RSS & RRD

These programs substitute for the actions taken by the MSHN Scheduler, the Resource Status Server, and the Resource Requirements Database. Each program begins execution in a wait status. Once the scheduler receives a job request from the Client, it performs the required security tests, pauses as a substitute for calculating a scheduling solution and returns the scheduling advice to the Client along with the required security token. The RSS and RRD programs wait for receipt of job statistics from the compute resource and after decryption/verification of these messages, store them in their respective databases.

I. SUMMARY

The MSHN security services prototype demonstrates that the MSHN_SL is a viable concept. It also verifies that CDSA is a workable solution for providing the necessary cryptographic mechanisms.

VII. RECOMMENDATIONS AND CONCLUSIONS

Clearly, if MSHN is to be successful, it must provide a level of trust and assurance consistent with the value of the information it is processing. This thesis demonstrates a security architecture for MSHN that is practical, flexible, and can be used to achieve the aforementioned goal.

A. RECOMMENDATIONS

The completion of the MSHN security services demonstration program is merely the starting point upon which further research can be conducted. Using this working model, we can now begin to perform statistical analysis on the overhead cost of security, in terms of computational speed.

The demonstration program is written specifically for personal computers running Windows NT. We recommend adapting the MSHN security layer to other platforms, such as UNIX. Also, the current demonstration program is written specifically for Intel's CDSA implementation. We recommend researching competitive vendor cryptographic products for comparison versus the Intel baseline.

Audit service is an essential component of the MSHN security-enhanced architecture. The current demonstration program does not implement this critical service. We

recommend further research on the design and implementation of an audit server for MSHN.

The inclusion of local proxy servers in the MSHN architecture may affect the security framework defined in this thesis. Their use was not considered during the security architecture design. Further research is required to determine the security implications of local proxies.

B. SUMMARY AND CONCLUSIONS

When MSHN is fully implemented, its ability to maximize the computational power of a complex, heterogeneous, network of resources will be a valuable tool. There appears to be no limit to users' appetite for automated decision support. Military and civilian decision makers alike demand and deserve the best information available, in a timely manner.

MSHN will be well suited to support command and control information systems. However, it is usable only to the extent that users have confidence that the security of their sensitive data will be preserved. MSHN must provide security mechanisms that are transparent and that take advantage of assurance in the underlying components of the VHM. We have concluded that the security services provided by the MSHN security layer can help achieve this objective.

APPENDIX A. MSHN SECURITY LAYER SOURCE CODE

A. MSHN_SL.H HEADER FILE

```
//****************************************************
//****************************************************
// File:  mshn_sl.h
// Name:  David Shifflett & Roger Wright
//
// Project: MSHN
//
// Operating Environment: Windows 95/Windows NT
// Compiler: Borland C++ for Windows
// Date:  18 MAY 98
//
// Description: Security services for MSHN components
//****************************************************

#ifndef _MSHN_SL_H
#define _MSHN_SL_H

#include "mshn_types.h"

// MSHN Security layer class

// Maximum number of open data stores
#define NUM_DS_HANDLES                      6

// MSHN Cryptographic identifiers
#define MSHN_NUM_ALG_IDS                    4
#define MSHN_ALG_DES                        1
#define MSHN_ALG_MD5                        2
#define MSHN_ALG_SHA                        3
#define MSHN_ALG_DSA                        4

// Cryptographic algorithm structure
typedef struct alg_params {
      int   mshn_id;
      int   alg_id;
      int   alg_mode;
      int   alg_padding;
} alg_params;

class mshn_sl {

// Class data
protected:
// Attached module handles
unsigned int      CL_handle;   // Cert Library module handle
```

```
unsigned int      CSP_handle;  // Crypto Services module handle
unsigned int      DL_handle;   // Data Store module handle
unsigned int      TP_handle;   // Trust Policy module handle

// context handles
int         sig_context;      // signature context
int         sym_context;      // symmetric enc/decrypt context
int         asym_context;     // asymmetric enc/decrypt context
int         dig_context;      // message digest context
int         ran_context;      // random encrypt/decrypt context
int         key_gen_context;  // key generation context

// open data store handles
unsigned int             ds_handles[NUM_DS_HANDLES];
char                    *ds_names[NUM_DS_HANDLES];

int         initialized;      // Has CSSM been initialized?

// array of crypto algorithm structures
alg_params          *alg_param_array;

public:
   mshn_sl();
   virtual ~mshn_sl();

// The Security Layer databases will be filled in based on
// the values in the configuration file.  The underlying
// security services will be "attached" based on the
// configuration values.  Any necessary security "contexts"
// will be created by this function.

int   mshn_sl_init (const char *config_file_name);

// This function will create a certificate with the provided
// parameters.
int   mshn_sl_create_cert (
            const char                  *issuername,
            const char                  *subjectname,
            const char                  *valid_from,
            const char                  *valid_to,
            int                          key_alg,
            const mshn_data             *public_key,
            mshn_data                   *new_cert);

// This function will find and return the certificate for
// the specified subject.
   int      mshn_sl_get_cert (
            const char                  *subjectname,
            mshn_data                   *the_cert);
```

```
// This function will determine if the specified certificate
// is valid.
   int       mshn_sl_cert_verify (
             const mshn_data      *the_cert,
             int                  *cert_valid);

// This function will determine if the specified certificate
// has been revoked.
   int       mshn_sl_cert_revoked (
             const mshn_data      *the_cert,
             int                  *cert_revoked);

// This function will return the public key for the subject
// associated with the specified certificate.
   int       mshn_sl_get_public_key (
             const mshn_data      *the_cert,
             mshn_data            *public_key);

// This function will return the private key for the
// specified subject.
   int       mshn_sl_get_private_key (
             const mshn_data      *public_key,
             mshn_data            *private_key);

// This function will write a record to the audit log.
   int       mshn_sl_put_audit (
             const char     *audit_record);

// This function will encrypt the specified buffers using
// the specified key and algorithm.  The caller of this
// function is responsible for deleting the memory returned
// by this function.
   int       mshn_sl_encrypt (
             const mshn_data  *the_key,
             int               the_alg,
             const mshn_data   buffer[],
             int               num_buffers,
             mshn_data        *enc_buff[],
             int              *num_encbuffs,
             int              *bytes_enc);

// This function will decrypt the specified buffers using
// the specified key and algorithm.  The caller of this
// function is responsible for deleting the memory returned
// by this function.
   int       mshn_sl_decrypt (
             const mshn_data  *the_key,
             int               the_alg,
             const mshn_data   enc_buff[],
             int               num_encbuffs,
             mshn_data        *buffer[],
```

```
        int                *num_buffers,
        int                *bytes_dec);

// This function will create a key to be used with symmetric
// encryption/decryption.
   int      mshn_sl_sym_key_gen (
            int                the_alg,
            int                key_size,
            const mshn_data    salt,
            const char        *key_label,
            mshn_data         *the_key);

// This function will create a public-private key pair to be
// used with asymmetric encryption/decryption.
   int      mshn_sl_asym_key_gen (
            int                the_alg,
            int                key_size,
            const mshn_data    salt,
            const char        *public_label,
            const char        *private_label,
            const char        *passphrase,
            mshn_data         *the_key);

// This function will hash the specified buffers and
// generate a message digest.
   int      mshn_sl_msg_digest (
            int                the_alg,
            const mshn_data    buffer[],
            int                num_buffers,
            mshn_data         *digest);

// This function will produce a signature for the specified
// data.  First the data will undergo a message digest
// operation.  The result of the message digest will then be
// encrypted to generate the signature.
   int      mshn_sl_sign (
            int                the_alg,
            int                hash_alg,
            const mshn_data   *the_key,
            const char        *password,
            const mshn_data    buffer[],
            int                num_buffers,
            mshn_data         *signature);

// This function will verify a signature on the specified
// data. First the data will undergo a message digest
// operation. The result of the message digest will be
// compared to the decryption of the specified signature.
   int      mshn_sl_sig_verify (
            int                the_alg,
            int                hash_alg,
```

```
            const mshn_data  *the_key,
            const mshn_data   buffer[],
            int               num_buffers,
            const mshn_data  *signature,
            int              *sig_valid);

// This function will return a character string description
// of the specified error code.  The caller of this function
// is responsible for deleting the memory returned by this
// function.
char *mshn_sl_show_error (
                  int                 the_error);

private:
// Get the error code from the underlying services
int   get_error();

// Find the chosen algorithm structure
int   find_alg_params (
            int    the_id,
            struct alg_params &the_params);

// Copy from the underlying key structure to a mshn_data
// structure
int   key_to_mshn_data (
            void        *key_ptr,
            mshn_data   *the_key);

// Copy from a mshn_data structure to the underlying key
// structure
int   mshn_data_to_key (
            const mshn_data  *the_key,
            void            **key_ptr);

};
#endif
```

B. MSHN_SL.CPP

```
//*******************************************************************
// File:  mshn_sl.cpp
// Name:  David Shifflett & Roger Wright
//
// Project: MSHN
//
// Operating Environment: Windows 95/Windows NT
// Compiler: Borland C++ for Windows
// Date: 18 MAY 98
//*******************************************************************
```

```
// Description: Security services for MSHN components
// MSHN Security layer class

#include <stdio.h>
#include <iostream.h>
#include <cssm.h>
#include <x509defs.h>
#include "mshn_sl.h"
#include "mshn_defs.h"
#include "mshn_err.h"
#include "mshn_mem.h"
#include "mshnUtil.h"
#include "showutil.h"

// constructor
mshn_sl::mshn_sl()
{
   CL_handle = CSSM_INVALID_HANDLE;
   CSP_handle = CSSM_INVALID_HANDLE;
   DL_handle = CSSM_INVALID_HANDLE;
   TP_handle = CSSM_INVALID_HANDLE;
   sig_context = CSSM_INVALID_HANDLE;
   sym_context = CSSM_INVALID_HANDLE;
   asym_context = CSSM_INVALID_HANDLE;
   dig_context = CSSM_INVALID_HANDLE;
   ran_context = CSSM_INVALID_HANDLE;
   key_gen_context = CSSM_INVALID_HANDLE;
   for (int idx=0; idx < NUM_DS_HANDLES;idx++) {
      ds_handles[idx] = CSSM_INVALID_HANDLE;
      ds_names[idx] = NULL;
   }
   initialized = MSHN_FALSE;
   alg_param_array = NULL;
}

// destructor
mshn_sl::~mshn_sl()
{
      if (alg_param_array != NULL) {
      mshn_free(alg_param_array, NULL);
      alg_param_array = NULL;
   }
   for (int idx=0; idx < NUM_DS_HANDLES;idx++) {
      if (ds_names[idx] != NULL) {
         mshn_free(ds_names[idx], NULL);
         ds_names[idx] = NULL;
      }
   }
}
```

```
//==============================================================
// The Security Layer databases will be filled in based on
// the values in the configuration file.  The underlying
// security services will be "attached" based on the
// configuration values.  Any necessary security "contexts"
// will be created by this function.
// Input:  Configuration file name.
// Output:  error code.
// Process:  This method performs initialization tasks as required by
// the underlying security API.  The mshn_sl_init method will read the
// configuration file specified by the input parameter and initialize
// the CSSM, and add-in modules according to the configuration values.
// An error code is returned if the initialization is unsuccessful.
// (For simplicity, the demonstration program uses a dummy file name.
// The actual initialization data is encoded as constants in the
// method.)

int mshn_sl::mshn_sl_init (
                  const char    *config_file_name)
{
   int result = MSHN_OK;

   if (initialized == MSHN_TRUE) {
      result = MSHN_INITIALIZED;
   }else{
      // initialize CSSM
      CSSM_RETURN retval = MSHN_CSSM_INIT();
      if (retval != CSSM_OK) {
            result = get_error();
      }else{
         // Now initialize the crypto algorithm structures
         // In the future, get this from a config file
         alg_param_array =
               (alg_params *) mshn_malloc(sizeof(alg_params)
               * MSHN_NUM_ALG_IDS, NULL);
         alg_param_array[0].mshn_id        = MSHN_ALG_DES;
         alg_param_array[0].alg_id         = CSSM_ALGID_DES;
         alg_param_array[0].alg_mode       = CSSM_ALGMODE_CBCPadIV8;
         alg_param_array[0].alg_padding    = CSSM_PADDING_PKCS5;
         alg_param_array[1].mshn_id        = MSHN_ALG_MD5;
         alg_param_array[1].alg_id         = CSSM_ALGID_MD5;
         alg_param_array[1].alg_mode       = 0;
         alg_param_array[1].alg_padding    = 0;
         alg_param_array[2].mshn_id        = MSHN_ALG_SHA;
         alg_param_array[2].alg_id         = CSSM_ALGID_SHA1WithDSA;
         alg_param_array[2].alg_mode       = 0;
         alg_param_array[2].alg_padding    = 0;
         alg_param_array[3].mshn_id = MSHN_ALG_DSA;
         alg_param_array[3].alg_id         = CSSM_ALGID_DSA;
         alg_param_array[3].alg_mode       = 0;
         alg_param_array[3].alg_padding    = 0;
```

```
        // Now attach the CSP, CL, DL modules
        CSSM_LIST_PTR pGUIDList;
        CSSM_MODULE_INFO_PTR pInfo;
        MSHN_CSP_INIT(pGUIDList, pInfo, CSP_handle);
        if (CSP_handle == CSSM_INVALID_HANDLE) {
              result = get_error();
        }else{
           MSHN_CL_INIT(pGUIDList, pInfo, CL_handle);
           if (CL_handle == CSSM_INVALID_HANDLE) {
                      result = get_error();
           }else{
              // This call will also attach a data store
              // For now just hard code datastore name
              char *hard_coded_name = "mshn";
              ds_names[0] = (char *)
                 mshn_malloc(strlen(hard_coded_name)+1, NULL);
              strcpy(ds_names[0], hard_coded_name);
              MSHN_DL_INIT(pGUIDList,
                                pInfo,
                                DL_handle,
                                ds_handles[0],
                                ds_names);
              if ((DL_handle == CSSM_INVALID_HANDLE) ||
                   (ds_handles[0] == CSSM_INVALID_HANDLE)) {
                              result = get_error();
              }else{
                 // For now ignore Trust Policy module

                 initialized = MSHN_TRUE;
              }
           }
        }
     }
   }
   return result;
}

//============================================================
// This function will create a certificate with the provided
// parameters.
// Input:  certificate fields.
// Output:  error code, pointer to a certificate.
// Process:  This method will create a certificate to include the
// public-private key pair associated with it.  The certificate created
// will contain the public key, while the private key is placed into a
// secure key storage facility.  An error code is returned if the
// initialization is unsuccessful.  (This method is not implemented in
// the demonstration program.  Instead, we used the certificate manager
// supplied by CDSA to create the certificates needed for the
// demonstration.)
```

```
int mshn_sl::mshn_sl_create_cert (
            const char          *issuername,
            const char          *subjectname,
            const char          *valid_from,
            const char          *valid_to,
            int                  key_alg,
            const mshn_data     *public_key,
            mshn_data           *new_cert)
{
   int result = MSHN_OK;

   if (initialized == MSHN_FALSE) {
      result = MSHN_NOT_INITIALIZED;
   }else{
      cout << "Function 'mshn_sl_create_cert' not implemented yet"
           << endl;
   }
   return result;
}

//=====================================================================
// This function will find and return the certificate for the specified
// subject.
// Input:  certificate subject name.
// Output:  error code, pointer to a certificate.
// Process:  This method will search the certificate database for a
// certificate whose subject name matches the input parameter and if
// found, returns the requested certificate.  The method uses CSSM Data
// library functions to query the database.  An error code is returned
// if the operation was unsuccessful.

int mshn_sl::mshn_sl_get_cert (
            const char                 *subjectname,
            mshn_data                  *the_cert)
{
   int result = MSHN_OK;
   the_cert->the_length = 0;
   the_cert->the_data = NULL;

   if (initialized == MSHN_FALSE) {
      result = MSHN_NOT_INITIALIZED;
   }else{

      CSSM_DL_DB_HANDLE dbHand;
      dbHand.DLHandle = DL_handle;
      dbHand.DBHandle = ds_handles[0];

      CSSM_HANDLE ResultsHandle = NULL;
      CSSM_QUERY Query;
      CSSM_BOOL EODS;
```

```
      CSSM_DATA_PTR certdata =
     (CSSM_DATA_PTR)mshn_malloc(sizeof(CSSM_DATA), NULL);
      CSSM_DB_UNIQUE_RECORD_PTR record_ptr;

      // Use a NULL filter to CSSM to get all certificates in database
      Query.NumSelectionPredicates = 0;
      Query.SelectionPredicate = NULL;
      Query.RecordType = CSSM_DL_DB_RECORD_CERT;
      Query.Conjunctive = CSSM_DB_NONE;
      record_ptr = CSSM_DL_DataGetFirst (dbHand,
                                         &Query,
                                         &ResultsHandle,
                                         &EODS,
                                          NULL,
                                          certdata);
#ifdef DEBUG_MSHN_CERT
      cout << "CSSM_DL_DataGetFirst done" << endl;
#endif
      show_error("CSSM_DL_DataGetFirst");
      CSSM_ClearError();

      // if end of data store before we even begin...
      if ((EODS == CSSM_TRUE) || (record_ptr == NULL)) {
         cout << "mshn_sl_get_cert: ERROR = Couldn't get first record"
              << endl;
      }else{
         int cntr = 1;
         int found = 0;
         while ((EODS == CSSM_FALSE) && !found)
            {
              // now find the certificate matching the subject
#ifdef DEBUG_MSHN_CERT
      cout << "get_cert_field" << endl;
#endif

            CSSM_DATA_PTR sdata = get_cert_field(
                                              certdata,
                                              CL_handle,
                                              CSSMOID_X509V1SubjectName);
#ifdef DEBUG_MSHN_CERT
      cout << "get_cert_field done" << endl;
#endif
            CSSM_DATA_PTR pstring;
            pstring = (CSSM_DATA_PTR)CSSM_CL_PassThrough (
                    CL_handle,
                    0,
                    INTEL_X509V3_PASSTHROUGH_TRANSLATE_DERNAME_TO_STRING,
                    sdata);

#ifdef DEBUG_MSHN_CERT
            show_data_char(*pstring, "trying subject name");
```

```
#endif
            CSSM_DL_FreeUniqueRecord(dbHand,record_ptr);

            if (match_field(pstring, subjectname, ";", 4)
                == CSSM_TRUE) {
               found = 1;
               the_cert->the_length = certdata->Length;
               the_cert->the_data = certdata->Data;
#ifdef DEBUG_MSHN_CERT
               show_cert_fields(certdata, CL_handle);
#endif
            }else{
               // Get the next certificate
               mshn_free(certdata->Data, NULL);
               record_ptr = CSSM_DL_DataGetNext (dbHand,
                                                  ResultsHandle,
                                                 &EODS,
                                                  NULL,
                                                  certdata);

               show_error("CSSM_DL_DataGetNext");
               CSSM_ClearError();
            }
            mshn_free(sdata->Data, NULL);
            mshn_free(pstring->Data, NULL);

         }
         if (!found)
            result = MSHN_CERT_NOT_FOUND;
         }
         mshn_free(certdata, NULL);
         // Done querying for information

         if (ResultsHandle)
            CSSM_DL_AbortQuery(dbHand, ResultsHandle);
   }
   return result;
}

//=====================================================================
// This function will determine if the specified certificate is valid.
// Input:  pointer to a certificate.
// Output:  error code, Boolean.
// Process:  This method will check the signature on a given certificate
// and return a Boolean value based upon the signature verification.
// (This method is not currently implemented in the demonstration
// program.  If called, it will always return true.)

int mshn_sl::mshn_sl_cert_verify (
            const mshn_data      *the_cert,
            int                  *cert_valid)
```

```
{
    int result = MSHN_OK;
   *cert_valid = MSHN_FALSE;

   if (initialized == MSHN_FALSE) {
      result = MSHN_NOT_INITIALIZED;
   }else{
      // For now don't bother verifying the certificate
      *cert_valid = MSHN_TRUE;
   }
   return result;
}

//=================================================================
// This function will determine if the specified certificate has been
// revoked.
// Input: pointer to a certificate.
// Output: error code, Boolean.
// Process:  This method will check a given certificate against a
// certificate revocation list and return a Boolean value depending on
// the revocation status. (This method is not currently implemented in
// the demonstration program.  If called, it will always return false.)

int mshn_sl::mshn_sl_cert_revoked (
            const mshn_data     *the_cert,
            int                 *cert_revoked)
{
    int result = MSHN_OK;
   *cert_revoked = MSHN_TRUE;

   if (initialized == MSHN_FALSE) {
      result = MSHN_NOT_INITIALIZED;
   }else{
      // For now don't bother checking the revocation list
      *cert_revoked = MSHN_FALSE;
   }
   return result;
}

//=================================================================
// This function will return the public key for the subject associated
// with the specified certificate.
// Input: pointer to a certificate.
// Output:  error code,  pointer to a public key.
// Process:  This method will return a public key obtained from a given
// certificate.  The method uses CSSM Certificate library functions to
// extract the key data.  An error code is returned if the operation was
// unsuccessful.

int mshn_sl::mshn_sl_get_public_key (
            const mshn_data     *the_cert,
```

```
            mshn_data           *public_key)
{
   int result = MSHN_OK;
   public_key->the_length = 0;
   public_key->the_data = NULL;

   if (initialized == MSHN_FALSE) {
      result = MSHN_NOT_INITIALIZED;
   }else{
      CSSM_DATA_PTR temp_cert =
     (CSSM_DATA_PTR)mshn_malloc(sizeof(CSSM_DATA), NULL);
      temp_cert->Length = the_cert->the_length;
      temp_cert->Data = the_cert->the_data;
#ifdef DEBUG_MSHN_CERT
      show_cert_fields(temp_cert, CL_handle);
#endif

      CSSM_KEY_PTR temp_key =
     (CSSM_KEY_PTR)mshn_malloc(sizeof(CSSM_KEY),NULL);
      temp_key = CSSM_CL_CertGetKeyInfo(CL_handle, temp_cert);
      show_error("CSSM_CL_CertGetKeyInfo");
      result = get_error();
      key_to_mshn_data(temp_key, public_key);
#ifdef DEBUG_MSHN_CERT
      show_key(temp_key, "Public Key");
#endif
      // Now clean up the allocated memory
      mshn_free(temp_cert, NULL);
      mshn_free(temp_key->KeyData.Data, NULL);
      mshn_free(temp_key, NULL);
   }
   return result;
}

//======================================================================
// This function will return the private key for the specified subject.
// Input:  pointer to a public key.
// Output:  error code, pointer to a private key.
// Process:  This method will return a reference to a private key given
// the associated public key.  The method uses CSSM Cryptographic
// Service Module functions to retrieve the key data from the CSP key
// storage file.  The private key is never openly exposed to the
// application program.  Instead, the pointer returned references an
// encrypted copy of the private key.  The private key can only be used
// when it is accompanied by its pass phrase.  An error code is returned
// if the operation was unsuccessful.

int mshn_sl::mshn_sl_get_private_key (
            const mshn_data     *public_key,
            mshn_data           *private_key)
{
```

```
   int result = MSHN_OK;
   private_key->the_length = 0;
   private_key->the_data = NULL;
   CSSM_KEY_PTR temp_key = NULL;

   if (initialized == MSHN_FALSE) {
      result = MSHN_NOT_INITIALIZED;
   }else{
      // convert inputs into CDSA structure
      void *temp_pub_key = NULL;

      mshn_data_to_key(public_key, &temp_pub_key);
#ifdef DEBUG_MSHN_KEY
      show_key((CSSM_KEY_PTR)temp_pub_key, "public key");
#endif
      temp_key = (CSSM_KEY_PTR)mshn_malloc(sizeof(CSSM_KEY),NULL);


// setting these fields to NULL will tell the CSP to allocate the
// memory for us
      temp_key->KeyData.Data =  NULL;
      temp_key->KeyData.Length = 0;

      CSSM_RETURN ok_key;
      ok_key = CSSM_CSP_ObtainPrivateKeyFromPublicKey(
               CSP_handle, (CSSM_KEY_PTR) temp_pub_key, temp_key);

      if (ok_key != CSSM_OK) {
         show_error("CSSM_CSP_ObtainPrivateKeyFromPublicKey");
         result = get_error();
      }else{
            key_to_mshn_data(temp_key, private_key);
#ifdef DEBUG_MSHN_KEY
      show_key((CSSM_KEY_PTR)temp_key, "private key");
#endif
      }
      // Now clean up the allocated memory
      CSSM_KEY_PTR pKey = (CSSM_KEY_PTR)temp_pub_key;
      mshn_free(pKey->KeyData.Data, NULL);
      mshn_free(pKey, NULL);
      mshn_free(temp_key->KeyData.Data, NULL);
      mshn_free(temp_key, NULL);
   }
   return result;
}

//=================================================================
// This function will write a record to the audit log.
// Input:  audit record.
// Output:  error code.
// Process:  This method will post a transaction to the MSHN audit
```

```
// server.  (The mshn_sl_put_audit method is not currently implemented
// in the demonstration program.)

int mshn_sl::mshn_sl_put_audit (
            const char    *audit_record)
{
      int result = MSHN_OK;

   if (initialized == MSHN_FALSE) {
      result = MSHN_NOT_INITIALIZED;
   }else{
      // For now just output the audit data
      cout << audit_record << endl;
   }
   return result;
}

//=================================================================
// This function will encrypt the specified buffers using the specified
// key and algorithm.  The caller of this function is responsible for
// deleting the memory returned by this function.
// Input:  pointers to a key, array of input buffers, and a count of the
// number of input buffers, and an algorithm identifier.
// Output:  error code, pointer to an array of encrypted buffers, a
// count of the number of encrypted buffers, and a count of the total
// number of bytes encrypted.
// Process:  This method will accept a data buffer and key, and return
// an encrypted copy of the input data. The method uses CSSM
// Cryptographic Service Module functions to encrypt the data based upon
// the key type and algorithm chosen.  This method supports both
// symmetric and asymmetric algorithms.  (Only symmetric encryption is
// operational in the demonstration program.  The Intel CSP lacks the
// implementation of a public key algorithm.)  An error code is returned
// if the operation was unsuccessful.

int mshn_sl::mshn_sl_encrypt (
            const mshn_data  *the_key,
            int               the_alg,
            const mshn_data   buffer[],
            int               num_buffers,
            mshn_data        *enc_buff[],
            int              *num_encbuffs,
            int              *bytes_enc)
{
   int result = MSHN_OK;
   struct alg_params the_params;
   unsigned int bytes_encrypted = 0;
   CSSM_CC_HANDLE hCC;
#if defined(DEBUG_MSHN_KEY) || defined(DEBUG_MSHN_ENCRYPT)
      char debug_in[80];
#endif
```

```
   for (int x=0; x<*num_encbuffs; x++) {
      enc_buff[x]->the_data = NULL;
      enc_buff[x]->the_length = 0;
   }
   *num_encbuffs = 0;
   *bytes_enc = 0;

   // make sure we have enough output buffers
   if (initialized == MSHN_FALSE) {
      result = MSHN_NOT_INITIALIZED;
   }else{
      // Get the algorithm parameters
      result = find_alg_params(the_alg, the_params);
   }
   if (result == MSHN_OK) {
      // convert inputs into CDSA structure
      void *temp_key = NULL;
      mshn_data_to_key(the_key, &temp_key);
#ifdef DEBUG_MSHN_KEY
      show_key((CSSM_KEY_PTR)temp_key, "encrypt key");
      cout << endl << "Press enter "; cin.getline(debug_in, 80);
#endif

#ifdef DEBUG_MSHN_ENCRYPT
      show_data_ptr((CSSM_DATA_PTR) buffer,
                    "encrypt input ", num_buffers);
      cout << endl << "Press enter "; cin.getline(debug_in, 80);
#endif
      // create encryption context
      hCC = CSSM_CSP_CreateSymmetricContext(CSP_handle,
                                            the_params.alg_id,
                                            the_params.alg_mode,
                                            (CSSM_KEY_PTR) temp_key,
                                            NULL,    // no initial vector
                                            the_params.alg_padding,
                                            0        // 0 rounds
                                            );
      if (hCC == 0) {
         cout << "Error creating encrypt context" << endl;
         show_error("CSSM_CSP_CreateSymmetricContext");
         result = get_error();
      }
      // Now clean up the allocated memory
      CSSM_KEY_PTR pKey = (CSSM_KEY_PTR)temp_key;
      mshn_free(pKey->KeyData.Data, NULL);
      mshn_free(pKey, NULL);
   }
   if (result == MSHN_OK) {
      CSSM_QUERY_SIZE_DATA queryData[10];
```

```
      for (int x = 0; x<num_buffers; x++) {
         queryData[x].SizeInputBlock = buffer[x].the_length;
         queryData[x].SizeOutputBlock = 0;
      }

      CSSM_QuerySize(hCC,
                     CSSM_TRUE,  // for encryption
                     num_buffers,
                     &queryData[0]);

      // allocate memory to hold the encrypted bits
      for (int x = 0; x<num_buffers; x++) {
         enc_buff[x]->the_length = queryData[x].SizeOutputBlock;
         enc_buff[x]->the_data = (unsigned char*)
         mshn_malloc(queryData[x].SizeOutputBlock,NULL);
#ifdef DEBUG_MSHN_ENCRYPT
         cout << "encrypt length: " << enc_buff[x]->the_length << endl;
#endif
      }

      unsigned int temp_encrypted = 0;
      CSSM_RETURN cssmstatus;
      CSSM_DATA remData;
      remData.Length = 0;
      remData.Data = 0;

      for (int x=0; ((x<num_buffers) && (result == MSHN_OK)); x++) {
         cssmstatus = CSSM_EncryptData(hCC,
                                       (CSSM_DATA_PTR) &buffer[x],
                                       1,        // number of input buffers
                                       (CSSM_DATA_PTR) enc_buff[x],
                                       1,        // number of enc buffers
                                       &temp_encrypted,
                                       &remData
                                       );
         bytes_encrypted += temp_encrypted;
#ifdef DEBUG_MSHN_ENCRYPT
         show_data_ptr((CSSM_DATA_PTR) enc_buff[x], "Encrypted", 1);
         cout << endl << "Press enter "; cin.getline(debug_in, 80);
#endif

         if(remData.Data != NULL) {
            cout << "rem data not null, rem data length: "
                 << remData.Length << endl;
            mshn_free(remData.Data,NULL);
         }

         if (cssmstatus != CSSM_OK) {
            result = get_error();
         }
      }
```

```
      CSSM_DeleteContext(hCC);
   }

   if (result == MSHN_OK)  {
      *num_encbuffs = num_buffers;
      *bytes_enc = bytes_encrypted;
   }
   return result;
}

//======================================================================
// This function will decrypt the specified buffers using the specified
// key and algorithm.  The caller of this function is responsible for
// deleting the memory returned by this function.
// Input:  pointers to a key, array of input buffers, and a count of the
// number of input buffers, and an algorithm code.
// Output:  error code, pointer to an array of decrypted buffers, a
// count of the number of decrypted buffers, and a count of the total
// number of bytes decrypted.
// Process:  This method will accept a data buffer and key, and return a
// decrypted copy of the input data. The method uses CSSM Cryptographic
// Service Module functions to decrypt the data based upon the key type
// and algorithm chosen.  This method supports both symmetric and
// asymmetric algorithms.  (Only symmetric decryption is operational in
// the demonstration program).  An error code is returned if the
// operation was unsuccessful.

int mshn_sl::mshn_sl_decrypt (
            const mshn_data  *the_key,
            int               the_alg,
            const mshn_data   enc_buff[],
            int               num_encbuffs,
            mshn_data        *buffer[],
            int              *num_buffers,
            int              *bytes_dec)
{

   int result = MSHN_OK;
   struct alg_params the_params;
   unsigned int bytes_decrypted = 0;
   CSSM_CC_HANDLE hCC;
#if defined(DEBUG_MSHN_KEY) || defined(DEBUG_MSHN_ENCRYPT)
      char debug_in[80];
#endif

   for (int x=0; x<*num_buffers; x++) {
      buffer[x]->the_data = NULL;
      buffer[x]->the_length = 0;
   }
   *num_buffers = 0;
   *bytes_dec = 0;
```

```
// make sure we have enough output buffers

   if (initialized == MSHN_FALSE) {
      result = MSHN_NOT_INITIALIZED;
   }else{
      // Get the algorithm parameters
      result = find_alg_params(the_alg, the_params);
   }
   if (result == MSHN_OK) {
      // convert inputs into CDSA structure
      void *temp_key = NULL;
      mshn_data_to_key(the_key, &temp_key);
#ifdef DEBUG_MSHN_KEY
      show_key((CSSM_KEY_PTR)temp_key, "encrypt key");
      cout << endl << "Press enter "; cin.getline(debug_in, 80);
#endif

#ifdef DEBUG_MSHN_DECRYPT
      show_data_ptr((CSSM_DATA_PTR)enc_buff,
                    "To Be Decrypted", num_encbuffs);
      cout << endl << "Press enter "; cin.getline(debug_in, 80);
#endif

      // create encryption context
      hCC = CSSM_CSP_CreateSymmetricContext(CSP_handle,
                                            the_params.alg_id,
                                            the_params.alg_mode,
                                            (CSSM_KEY_PTR) temp_key,
                                            NULL,    // no initial vector
                                            the_params.alg_padding,
                                            0        // 0 rounds
                                            );
      if (hCC == 0)
      {
         cout << "Error creating decrypt context" << endl;
         show_error("CSSM_CSP_CreateSymmetricContext");
         result = get_error();
      }
      // Now clean up the allocated memory
      CSSM_KEY_PTR pKey = (CSSM_KEY_PTR)temp_key;
      mshn_free(pKey->KeyData.Data, NULL);
      mshn_free(pKey, NULL);
   }
   if (result == MSHN_OK) {
      // this is the return value
      CSSM_QUERY_SIZE_DATA queryData[10];

      for (int x = 0; x<num_encbuffs; x++) {
         queryData[x].SizeInputBlock = enc_buff[x].the_length;
         queryData[x].SizeOutputBlock = 0;
```

```
    }

    CSSM_QuerySize(hCC,
                      CSSM_FALSE,  // for decryption
                      num_encbuffs,
                      &queryData[0]);

    // allocate memory to hold the encrypted bits
    for (int x = 0; x<num_encbuffs; x++) {
       buffer[x]->the_length = queryData[x].SizeOutputBlock;
       buffer[x]->the_data = (unsigned char*)
       mshn_malloc(queryData[x].SizeOutputBlock,NULL);
#ifdef DEBUG_MSHN_DECRYPT
       cout << "decrypt length: " << buffer[x]->the_length << endl;
#endif
     }

    unsigned int temp_decrypted = 0;
    CSSM_RETURN cssmstatus;
    CSSM_DATA remData;
    remData.Length = 0;
    remData.Data = 0;

    for (int x=0; ((x<num_encbuffs) && (result == MSHN_OK)); x++) {
       cssmstatus = CSSM_DecryptData(hCC,
                                     (CSSM_DATA_PTR) &enc_buff[x],
                                     1,        // number of input buffers
                                     (CSSM_DATA_PTR) buffer[x],
                                     1,        // number of dec buffers
                                     &temp_decrypted,
                                     &remData
                                     );
       bytes_decrypted += temp_decrypted;
#ifdef DEBUG_MSHN_DECRYPT
       show_data_ptr((CSSM_DATA_PTR) buffer[x], "Decrypted", 1);
       cout << endl << "Press enter "; cin.getline(debug_in, 80);
#endif

       if(remData.Data != NULL) {
          cout << "rem data not null " << "rem data length: "
               << remData.Length <<endl;
          mshn_free(remData.Data,NULL);
       }

       if (cssmstatus != CSSM_OK) {
          show_error("CSSM Decrypt: ");
          result = get_error();
       }
    }
       CSSM_DeleteContext(hCC);
    }
```

```
   if (result == MSHN_OK)   {
      *bytes_dec = bytes_decrypted;
      *num_buffers = num_encbuffs;
   }
   return result;
}

//==================================================================
// This function will create a key to be used with symmetric
// encryption/decryption.
// Input:  algorithm code, key size, salt, and key label.
// Output:  error code, pointer to a key.
// Process:  This method generates a symmetric key. It uses CSSM
// Cryptographic Service Module functions to create the key data.  The
// salt parameter can be used to effectively expand the key size.  An
// error code is returned if the operation was unsuccessful.

int mshn_sl::mshn_sl_sym_key_gen (
            int                 the_alg,
            int                 key_size,
            const mshn_data     salt,
            const char         *key_label,
            mshn_data          *the_key)
{
   int result = MSHN_OK;
   struct alg_params the_params;
   the_key->the_data = NULL;
   the_key->the_length = 0;

   if (initialized == MSHN_FALSE) {
      result = MSHN_NOT_INITIALIZED;
   }else{
      // Get the algorithm parameters
      result = find_alg_params(the_alg, the_params);
   }

   CSSM_RETURN cssmstatus;
   CSSM_CC_HANDLE hCC = NULL;

   if (result == MSHN_OK) {
         hCC = CSSM_CSP_CreateKeyGenContext(
                  CSP_handle,  // CSP handle
                  the_params.alg_id,
                  NULL,        // pass phrase not req for DES
                  key_size,    // key size
                  NULL,        // seed
                  NULL,        // salt
                  NULL,        // start date
                  NULL,        // end date
                  NULL);       // params
```

```
      if(hCC == NULL)
         {
               cout << "Error creating key generation context" << endl;
               show_error("CSSM_CSP_CreateKeyGenContext");
               result = get_error();
         }
   }

   if (result == MSHN_OK) {
      CSSM_KEY_PTR pKey =
     (CSSM_KEY_PTR)mshn_malloc(sizeof(CSSM_KEY),NULL);

      // setting these fields to NULL will tell the CSP to allocate
      // the memory for us
      pKey->KeyData.Data =  NULL;
      pKey->KeyData.Length = 0;

      CSSM_DATA key_lab;
      key_lab.Length = strlen(key_label);
#ifdef DEBUG_MSHN_KEY
      cout << "key label length: " << key_lab.Length << endl;
#endif
      key_lab.Data = (unsigned char*)mshn_malloc(key_lab.Length, NULL);
      strncpy(key_lab.Data, key_label, key_lab.Length);

      cssmstatus = CSSM_GenerateKey(hCC,  // context handle
                            CSSM_KEYUSE_ANY, // usage
                            CSSM_KEYATTR_RETURN_DEFAULT, // attributes
                           &key_lab, // label
                            pKey); // the key

      if(cssmstatus != CSSM_OK)
      {
         cout << "Error creating CSSM key" << endl;
         show_error("CSSM_GenerateKey");
         result = get_error();
      }else{
         key_to_mshn_data(pKey, the_key);
#ifdef DEBUG_MSHN_KEY
         show_key(pKey, "DES Key");
         void *temp_key = NULL;
         mshn_data_to_key(the_key, &temp_key);
         show_key((CSSM_KEY_PTR)temp_key, "extracted key");
         // Now clean up the allocated memory
         CSSM_KEY_PTR tpKey = (CSSM_KEY_PTR)temp_key;
         mshn_free(tpKey->KeyData.Data, NULL);
         mshn_free(tpKey, NULL);
#endif
      }
      // Now clean up the allocated memory
```

```
      mshn_free(key_lab.Data, NULL);
      mshn_free(pKey->KeyData.Data, NULL);
      mshn_free(pKey, NULL);

      CSSM_DeleteContext(hCC);
   }
   return result;
}

//=====================================================================
// This function will create a public-private key pair to be used with
// asymmetric encryption/decryption.
// Input:  algorithm code, key size, salt, key labels, and pass phrase
// for the private key.
// Output:  error code, pointer to a key pair.
// Process:  This method generates an asymmetric (public/private) key
// pair.  The private key is placed in a secure key storage facility.
// (The mshn_sl_asym_key_gen method is not implemented in the
// demonstration program.  We used the certificate manager that came
// with CDSA to create the certificates needed for the demonstration.
// Certificate manager generated the public/private key pairs as part of
// the certificate creation process.)

int mshn_sl::mshn_sl_asym_key_gen (
            int                  the_alg,
            int                  key_size,
            const mshn_data      salt,
            const char           *public_label,
            const char           *private_label,
            const char           *passphrase,
            mshn_data            *the_key)
{
   int result = MSHN_OK;

   if (initialized == MSHN_FALSE) {
      result = MSHN_NOT_INITIALIZED;
   }else{
      cout << "Function 'mshn_sl_asym_key_gen' not implemented yet"
           << endl;
   }
   return result;
}

//======================================================================
// This function will hash the specified buffers and generate a message
// digest.
// Input: pointer to an array of input buffers, and a count of the
// number of input buffers, and an algorithm identifier.
// Output:  error code, pointer to a digest structure.
// Process:  This method creates a message digest for a given input
// buffer of data. The method uses CSSM Cryptographic Service Module
```

```
// functions to generate the digest, based upon the algorithm chosen.
// The demonstration program uses the MD5 algorithm to create a fixed
// size, 16 byte digest from the given data.  An error code is returned
// if the operation was unsuccessful.

int mshn_sl::mshn_sl_msg_digest (
            int                 the_alg,
            const mshn_data     buffer[],
            int                 num_buffers,
            mshn_data           *digest)
{
   int result = MSHN_OK;
   digest->the_data = NULL;
   digest->the_length = 0;
   struct alg_params the_params;

   if (initialized == MSHN_FALSE) {
      result = MSHN_NOT_INITIALIZED;
   }
   else {
      // Get the algorithm parameters
      result = find_alg_params(the_alg, the_params);

      CSSM_RETURN cssmstatus;
      CSSM_CC_HANDLE hdigestContext;
      // this is the return value
      CSSM_DATA_PTR pDig =
     (CSSM_DATA_PTR)mshn_malloc(sizeof(CSSM_DATA),NULL);
      pDig->Data = NULL;
      pDig->Length = 0;

      if (result == MSHN_OK) {
         hdigestContext = CSSM_CSP_CreateDigestContext(
                  CSP_handle, the_params.alg_id);

         if (hdigestContext == 0)
            {
            cout << "Error creating digest context" << endl;
            show_error("CSSM_CSP_CreateDigestContext");
            result = get_error();
            }
      }

      if (result == MSHN_OK) {
            cssmstatus = CSSM_DigestData(hdigestContext,
                              (CSSM_DATA_PTR) buffer,
                              num_buffers,
                              pDig);

            if (cssmstatus != CSSM_OK) {
                  cout << "Digest creation failed" << endl;
```

```
                    show_error("Digest Error");
                    result = get_error();
              }else{
#ifdef DEBUG_MSHN_SIGN
                 cout << "\nDigest Size: " << pDig->Length << endl;
#endif
                 digest->the_data = pDig->Data;
                 digest->the_length = pDig->Length;
              }

              CSSM_DeleteContext(hdigestContext);
        }
        mshn_free(pDig, NULL);
   }
   return result;
}

//================================================================
// This function will produce a signature for the specified data.
// First the data will undergo a message digest operation.
// The result of the message digest will then be encrypted to generate
// the signature.
// Input:  Signature algorithm identifier, hash algorithm identifier,
// pointers to the key, key passphrase, and input data buffers, along
// with a count of the number of buffers provided.
// Output:  error code, pointer to a signature structure.
// Process:  This method creates a digital signature for a given buffer
// of data.  This method supports the creation of signatures using
// symmetric and asymmetric keys.  For example, if the signature
// algorithm identifier is DES, and hash algorithm identifier is MD5,
// then the method will create an MD5 digest of the data, and encrypt
// the digest with the supplied DES key.  If the signature algorithm
// identifier is DSA (Digital Signature Algorithm), and the hash
// algorithm identifier is SHA (Secure Hash Algorithm), the method will
// create an SHA digest of the data and encrypt the digest with the
// supplied private key. An error code is returned if the operation was
// unsuccessful.

int mshn_sl::mshn_sl_sign (
            int                  the_alg,
            int                  hash_alg,
            const mshn_data     *the_key,
            const char          *password,
            const mshn_data      buffer[],
            int                  num_buffers,
            mshn_data           *signature)
{
   int result = MSHN_OK;
   int alg_result = MSHN_OK;
   int hash_result = MSHN_OK;
   signature->the_data = NULL;
```

```
    signature->the_length = 0;

    struct alg_params the_alg_params, hash_params;
    void *pKey;
    mshn_data_to_key(the_key, &pKey);

    CSSM_RETURN cssmstatus;
    CSSM_CC_HANDLE hSigContext;
    CSSM_DATA_PTR pSig =
   (CSSM_DATA_PTR)mshn_malloc(sizeof(CSSM_DATA),NULL);
    CSSM_CRYPTO_DATA cspData;
    CSSM_DATA paramData;

    mshn_data *encSigBuff[1];
    encSigBuff[0] = (mshn_data *)mshn_malloc(sizeof(mshn_data *), NULL);

    int  bytesEnc = 0;
    int  numEncBuffs = 1;

    if (initialized == MSHN_FALSE) {
       result = MSHN_NOT_INITIALIZED;
    }
    else
    {
       // Get the algorithm parameters
       alg_result = find_alg_params(the_alg, the_alg_params);
       hash_result = find_alg_params(hash_alg, hash_params);

       if (alg_result == MSHN_OK && hash_result == MSHN_OK) {

       switch (the_alg_params.mshn_id) {

          case MSHN_ALG_DSA:  { // DSA Algorithm

                   // create signature context

                   // this is the return value
                         pSig->Data = NULL;
                         pSig->Length = 0;

                   // Set up the crypto data
                         cspData.Callback = NULL;

                   // The "cspData" is the password for the signer's
                   // private key
#ifdef DEBUG_MSHN_SIGN
                   cout << " password length " << strlen(password)
                        << endl;
#endif

                   if (strlen(password))
```

```
                {
                   paramData.Length = strlen(password);
                   paramData.Data =
                   (uint8*)mshn_malloc(paramData.Length,NULL);
                   memcpy(paramData.Data,
                          password, paramData.Length);
                } else {
                         paramData.Length = 0;
                         paramData.Data = NULL;
                }
             cspData.Param = &paramData;
#ifdef DEBUG_MSHN_SIGN
             cout << "sig context password length: "
                  << paramData.Length << endl;
#endif

             hSigContext = CSSM_CSP_CreateSignatureContext(
                                 CSP_handle,
                                 hash_params.alg_id,
                                 &cspData,
                                 (CSSM_KEY_PTR) pKey);

             if (hSigContext == 0)
             {
                cout << "Error creating sig context" << endl;
                show_error("CSSM_CSP_CreateSignatureContext");
                result = get_error();
             }
       // Now cleanup allocated data
       mshn_free(paramData.Data, NULL);

       if (result == MSHN_OK) {
          cssmstatus = CSSM_SignData(hSigContext,
                                 (CSSM_DATA_PTR) buffer,
                                 num_buffers,
                                 pSig);

#ifdef DEBUG_MSHN_SIGN
          cout << "Signature size: " << pSig->Length << endl;
#endif

          if (cssmstatus != CSSM_OK) {
             show_error("Signature failed");
             result = get_error();
             mshn_free(pSig->Data, NULL);
          }else{
             signature->the_data = pSig->Data;
             signature->the_length = pSig->Length;
          }

          CSSM_DeleteContext(hSigContext);
```

```
            }
            break;
         }
         case MSHN_ALG_DES:   { // DES Algorithm
            mshn_data sig_digest;
            result = mshn_sl_msg_digest(
                           hash_alg,
                           buffer,
                           num_buffers,
                           &sig_digest);

            if (result == MSHN_OK) {
#ifdef DEBUG_MSHN_SIGN
            show_pointer((uint8 *)sig_digest.the_data,
                              sig_digest.the_length,
                             "Msg Digest");
#endif

            result = mshn_sl_encrypt(the_key,
                                    the_alg,
                                   &sig_digest,
                                    1,        // num input buffers
                                    encSigBuff,
                                   &numEncBuffs, // num output buffers
                                   &bytesEnc);
            }
            // Now cleanup allocated data
            mshn_free(sig_digest.the_data, NULL);

            if (result == MSHN_OK) {
#ifdef DEBUG_MSHN_SIGN
               show_pointer((uint8 *)encSigBuff[0]->the_data,
                                    encSigBuff[0]->the_length,
                                   "Encrypted msg Digest");
#endif
               signature->the_data = encSigBuff[0]->the_data;
               signature->the_length = encSigBuff[0]->the_length;
            }
            break;
         }
         default: // Invalid Signature algorithm
            cout << "Invalid Signature Algorithm" << endl;
        }
      }
   }
   // Now clean up the allocated memory
   CSSM_KEY_PTR tpKey = (CSSM_KEY_PTR)pKey;
   mshn_free(tpKey->KeyData.Data, NULL);
   mshn_free(tpKey, NULL);
   mshn_free(pSig, NULL);
   mshn_free(encSigBuff[0], NULL);
```

```
   return result;
}

//=====================================================================
// This function will verify a signature on the specified data.
// First the data will undergo a message digest operation.
// The result of the message digest will be compared to the decryption
// of the specified signature.
// Input: Signature algorithm code, hash algorithm code, pointers to the
// key, input data buffers, and signature, along with a count of the
// number of buffers provided.
// Output:  error code, verification result.
// Process:  This method accepts a signature and data buffer.  It
// returns a Boolean value depending on the verification of the
// signature against the input data.  If a symmetric signature algorithm
// was used, the method will decrypt the signature using the supplied
// key.  Then the method will generate a digest of the input data and
// compare the new digest to the decrypted signature.  If they match,
// the method returns true.  If an asymmetric algorithm was specified,
// then the signature is similarly verified using the supplied public
// key. An error code is returned if the operation was unsuccessful.

int mshn_sl::mshn_sl_sig_verify (
            int                 the_alg,
            int                 hash_alg,
            const mshn_data    *the_key,
            const mshn_data     buffer[],
            int                 num_buffers,
            const mshn_data    *signature,
            int                *sig_valid)
{
   int result = MSHN_OK;
   int alg_result = MSHN_OK;
   int hash_result = MSHN_OK;
   *sig_valid = MSHN_FALSE;
   struct alg_params the_alg_params, hash_params;
   void *pKey;
   mshn_data_to_key(the_key, &pKey);

   CSSM_BOOL cssmstatus;
   CSSM_CC_HANDLE hVerifContext;

   mshn_data *decSigBuff[1];
   decSigBuff[0] = (mshn_data *)mshn_malloc(sizeof(mshn_data *),NULL);
   int  bytesDec = 0;
   int  numDecBuffs = 1;
   void *decrypted_sig = NULL;
   mshn_data *digest;
   digest = (mshn_data *)mshn_malloc(sizeof(mshn_data), NULL);

   if (initialized == MSHN_FALSE) {
```

```
        result = MSHN_NOT_INITIALIZED;
    }
    else {
        // Get the algorithm parameters
        alg_result = find_alg_params(the_alg, the_alg_params);
        hash_result = find_alg_params(hash_alg, hash_params);
    }

    if (alg_result == MSHN_OK && hash_result == MSHN_OK) {

        switch (the_alg_params.mshn_id) {

            case MSHN_ALG_DSA:  {// DSA Algorithm

                fix_key_size((CSSM_KEY_PTR) pKey);
                hVerifContext = CSSM_CSP_CreateSignatureContext(
                                    CSP_handle,
                                    hash_params.alg_id,
                                    NULL,    // pass phrase not needed
                                    (CSSM_KEY_PTR) pKey);

                if (hVerifContext == 0)
                {
                        cout << "Error creating signature verification
                        context" << endl;
                        show_error("CSSM_CSP_CreateSignatureContext");
                        result = get_error();
                }

                if (result == MSHN_OK) {
                        cssmstatus = CSSM_VerifyData(hVerifContext,
                                    (CSSM_DATA_PTR) buffer,
                                    num_buffers,
                                    (CSSM_DATA_PTR) signature);

                if (cssmstatus != CSSM_TRUE) {
#ifdef DEBUG_MSHN_SIGN
                    cout << "\nVerification failed" << endl;
#endif
                   *sig_valid = MSHN_FALSE;
                }else{
#ifdef DEBUG_MSHN_SIGN
                    cout << "\nSignature matches public key" << endl;
#endif
                   *sig_valid = MSHN_TRUE;
                }
                CSSM_DeleteContext(hVerifContext);
                }
                break;
            }
            case MSHN_ALG_DES:  {  // DES Algorithm
```

```
#ifdef DEBUG_MSHN_SIGN
            show_pointer((uint8 *)signature->the_data,
                                  signature->the_length,
                                  "Verify Encrypted msg Digest");
#endif

            result = mshn_sl_decrypt(the_key,
                                   the_alg,
                                   signature,
                                   1,    // number of input buffers
                                   decSigBuff,
                                   &numDecBuffs, // num output buffers
                                   &bytesDec);

            if (result == MSHN_OK) {
#ifdef DEBUG_MSHN_SIGN
                show_pointer((uint8 *)decSigBuff[0]->the_data,
                                       decSigBuff[0]->the_length,
                                      "Decrypted msg Digest");
#endif
                result = mshn_sl_msg_digest(
                              hash_alg,
                              buffer,
                              num_buffers,
                              digest);
            }

            if (result == MSHN_OK) {
#ifdef DEBUG_MSHN_SIGN
                show_pointer((uint8 *)digest->the_data,
                                       digest->the_length,
                                       "Msg Digest");
#endif
                if (!memcmp(digest->the_data,
                            decSigBuff[0]->the_data,
                            digest->the_length)) {
#ifdef DEBUG_MSHN_SIGN
                    cout << "Signature Successfully Verified" << endl;
#endif
                    *sig_valid = MSHN_TRUE;
                    }else{
#ifdef DEBUG_MSHN_SIGN
                    cout << "Signature Verification Failed" << endl;
#endif
                    *sig_valid = MSHN_FALSE;
                    }
            }
            mshn_free(decSigBuff[0]->the_data, NULL);
            mshn_free(digest->the_data, NULL);

            break;
```

```
            }
            default: // Invalid Signature algorithm
               cout << "Invalid Signature Algorithm" << endl;
        }
    }
    // Now clean up the allocated memory
    CSSM_KEY_PTR tpKey = (CSSM_KEY_PTR)pKey;
    mshn_free(tpKey->KeyData.Data, NULL);
    mshn_free(tpKey, NULL);
    mshn_free(decSigBuff[0], NULL);
    mshn_free(digest, NULL);
    return result;
}

//=======================================================================
// This function will return a character string description
// of the specified error code.
// The caller of this function is responsible for deleting
// the memory returned by this function.
char *mshn_sl::mshn_sl_show_error (
                  int                the_error)
{
    char *result;
    switch (the_error) {
        case MSHN_OK: {
                result = strdup("No error");
                break;
        }
        case MSHN_NOT_INITIALIZED: {
                result = strdup("MSHN SL not initialized");
                break;
        }
        case MSHN_INITIALIZED: {
                result = strdup("MSHN SL already initialized");
                break;
        }
        case MSHN_ALG_NOT_FOUND: {
                result = strdup("Algorithm not found");
                break;
        }
        case MSHN_CERT_NOT_FOUND: {
                result = strdup("Certificate not found");
                break;
        }
        case MSHN_CERT_INVALID: {
                result = strdup("Certificate is invalid");
                break;
        }
        case MSHN_CERT_REVOKED: {
                result = strdup("Certificate has been revoked");
                break;
```

```
      }
      case MSHN_INVALID_SIG: {
            result = strdup("Signature is invalid");
            break;
      }
      default: {
            result = (char *)mshn_malloc(80, NULL);
            sprintf(result, "Unknown error (%d)", the_error);
            break;
      }
   }
   return result;
}

//=====================================================================
// This function will return the error code from the underlying
// security services provider.
int mshn_sl::get_error ()
{
   int result = MSHN_UNKNOWN_ERROR;
   CSSM_ERROR_PTR the_error = CSSM_GetError();
   if (the_error != NULL) {
      if (the_error->error == CSSM_OK) {
            result = MSHN_OK;
      }else{
            result = the_error->error;
      }
   }
   return result;
}

//=====================================================================
// Find the chosen algorithm structure
int mshn_sl::find_alg_params (
                  int the_id,
            struct alg_params &the_params)
{
   int result;
   if (initialized == MSHN_FALSE) {
      result = MSHN_NOT_INITIALIZED;
   }else{
      result = MSHN_ALG_NOT_FOUND;
      for (int idx=0; idx < MSHN_NUM_ALG_IDS; idx++) {
         if (alg_param_array[idx].mshn_id == the_id) {
            the_params.mshn_id = alg_param_array[idx].mshn_id;
            the_params.alg_id = alg_param_array[idx].alg_id;
            the_params.alg_mode = alg_param_array[idx].alg_mode;
            the_params.alg_padding = alg_param_array[idx].alg_padding;
            result = MSHN_OK;
            break;
         }
```

```
      }
   }
   return result;
}

//==========================================================================
// Copy from a CSSM_KEY structure to a mshn_data structure
int mshn_sl::key_to_mshn_data (
            void        *key_ptr,
            mshn_data  *the_key)
{
   int result = 0;
   CSSM_KEY_PTR temp_key = (CSSM_KEY_PTR)key_ptr;
   the_key->the_length = sizeof(CSSM_KEYHEADER)
                       + temp_key->KeyData.Length;
   the_key->the_data = (unsigned char *)
                        mshn_malloc(the_key->the_length, NULL);
   memcpy(the_key->the_data,
        &(temp_key->KeyHeader), sizeof(CSSM_KEYHEADER));
   memcpy(the_key->the_data + sizeof(CSSM_KEYHEADER),
        temp_key->KeyData.Data,
        temp_key->KeyData.Length);
   return result;
}

//==========================================================================
// Copy from a mshn_data structure to a CSSM_KEY structure
int mshn_sl::mshn_data_to_key (
            const mshn_data  *the_key,
            void             **key_ptr)
{
   int result = 0;
   CSSM_KEY_PTR temp_key = (CSSM_KEY_PTR) mshn_malloc(
                             sizeof(CSSM_KEY), NULL);
   temp_key->KeyData.Length = the_key->the_length
                               sizeof(CSSM_KEYHEADER);
   temp_key->KeyData.Data = (uint8 *)mshn_malloc(
                                   temp_key->KeyData.Length, NULL);
   memcpy(&(temp_key->KeyHeader),
            the_key->the_data, sizeof(CSSM_KEYHEADER));
   memcpy(temp_key->KeyData.Data, the_key->the_data
        + sizeof(CSSM_KEYHEADER), temp_key->KeyData.Length);
   *key_ptr = temp_key;
   return result;
}
```

APPENDIX B. MSHN DEMONSTRATION SOURCE CODE

A. CLIENT.CPP

```
//****************************************************
// File:  client.cpp
// Name:  Roger Wright
//
// Project: MSHN
//
// Operating Environment: Windows 95/Windows NT
// Compiler: Borland C++ for Windows
// Date: 12 MAY 98
//
// Description: MSHN demonstration client shell
//****************************************************

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <fstream.h>
#include "mshn_sl.h"
#include "mshn_mem.h"
#include "mshn_com.h"
#include "mshn_err.h"
#include "mshn_types.h"
#include "mshn_defs.h"
#include "mshn_demo.h"
#include "showutil.h"
#include "commutil.h"

const int BUFF_SIZE = 1024;

char *err_out;
mshn_sl  *msl_obj;
mshn_com *mc_obj;
mshn_data sym_key;

//==================================================================
// verify data that was received from the scheduler
int   do_verify(mshn_data resource_id,
                mshn_data job_info,
                mshn_data token_data,
                mshn_data job_sess_key,
                mshn_data core_cert,
                mshn_data sched_sig,
                int       &sig_valid)
{
```

```
   int result = MSHN_OK;

   // Now prepare the inputs for verification
   const int num_signed_params = 5;
   const int num_params = 6;
   mshn_data work_array[num_params];
   work_array[0].the_length = resource_id.the_length;
   work_array[0].the_data   = resource_id.the_data;
   work_array[1].the_length = job_info.the_length;
   work_array[1].the_data   = job_info.the_data;
   work_array[2].the_length = token_data.the_length;
   work_array[2].the_data   = token_data.the_data;
   work_array[3].the_length = job_sess_key.the_length;
   work_array[3].the_data   = job_sess_key.the_data;
   work_array[4].the_length = core_cert.the_length;
   work_array[4].the_data   = core_cert.the_data;
   work_array[5].the_length = sched_sig.the_length;
   work_array[5].the_data   = sched_sig.the_data;

   mshn_data public_key;

   result = msl_obj->mshn_sl_get_public_key(&core_cert, &public_key);

   if (result != MSHN_OK) {
      err_out = msl_obj->mshn_sl_show_error(result);
      cout << "mshn_sl_get_public_key " << err_out << endl;
      mshn_free(err_out, NULL);
   }

   else {

      result = msl_obj->mshn_sl_sig_verify(MSHN_ALG_DSA,
                                           MSHN_ALG_SHA,
                                          &public_key,
                                           work_array,
                                           num_signed_params,
                                          &work_array[5],    // the sig
                                           &sig_valid);

      if (result != MSHN_OK) {
               err_out = msl_obj->mshn_sl_show_error(result);
               cout << "mshn_sl_sig_verify " << err_out << endl;
               mshn_free(err_out, NULL);
      }
      if (!sig_valid) {
         result = MSHN_INVALID_SIG;
      }
   }

   return result;
}
```

```
//=====================================================================
// verify data that came from the compute resource
int    do_verify_resource(mshn_data results,
                          mshn_data app_signature,
                          mshn_data job_sess_key,
                          int       &sig_valid)
{
   int result = MSHN_OK;

   // Now prepare the inputs for verification
   const int num_signed_params = 1;
   const int num_params = 2;
   mshn_data work_array[num_params];
   work_array[0].the_length = results.the_length;
   work_array[0].the_data   = results.the_data;
   work_array[1].the_length = app_signature.the_length;
   work_array[1].the_data   = app_signature.the_data;

   result = msl_obj->mshn_sl_sig_verify(MSHN_ALG_DES,
                                        MSHN_ALG_MD5,
                                        &job_sess_key,
                                        work_array,
                                        num_signed_params,
                                        &work_array[1],     // the sig
                                        &sig_valid);

   if (result != MSHN_OK) {
         err_out = msl_obj->mshn_sl_show_error(result);
         cout << "mshn_sl_sig_verify " << err_out << endl;
         mshn_free(err_out, NULL);
   }

   if (!sig_valid) {
      result = MSHN_INVALID_SIG;
   }
   return result;
}

//=====================================================================
// sign data that will be transmitted to the scheduler
int    do_signature(
                    const mshn_data   *user_id,
                    const mshn_data   *cert,
                    const mshn_data   *sched_info,
                    const char        *passphrase,
                    mshn_data         *the_sig)
{
   int result = MSHN_OK;
   // Clear the output
   the_sig->the_length = 0;
   the_sig->the_data = NULL;
```

```
   // Now prepare the inputs for signing
   const int num_sign = 3;
   mshn_data work_array[num_sign];
   work_array[0].the_length = user_id->the_length;
   work_array[0].the_data   = user_id->the_data;
   work_array[1].the_length = cert->the_length;
   work_array[1].the_data   = cert->the_data;
   work_array[2].the_length = sched_info->the_length;
   work_array[2].the_data   = sched_info->the_data;

   mshn_data public_key;
   mshn_data private_key;
   result = msl_obj->mshn_sl_get_public_key(cert, &public_key);

   if (result != MSHN_OK) {
      err_out = msl_obj->mshn_sl_show_error(result);
      cout << "mshn_sl_get_public_key " << err_out << endl;
      mshn_free(err_out, NULL);
      }
   else{
      result = msl_obj->mshn_sl_get_private_key(
                        &public_key, &private_key);

      if (result != MSHN_OK) {
            err_out = msl_obj->mshn_sl_show_error(result);
            cout << "mshn_sl_get_private_key " << err_out << endl;
            mshn_free(err_out, NULL);
      }
   }
   if (result == MSHN_OK) {
      result = msl_obj->mshn_sl_sign(MSHN_ALG_DSA,
                                     MSHN_ALG_SHA,
                                     &private_key,
                                     passphrase,
                                     work_array,
                                     num_sign,
                                     the_sig);

      if (result != MSHN_OK) {
            err_out = msl_obj->mshn_sl_show_error(result);
            cout << "mshn_sl_sign " << err_out << endl;
            mshn_free(err_out, NULL);
      }
  }
   return result;
}
```

```
//======================================================================
// sign data going to the compute resource
int   do_signature_resource(
                  const mshn_data    *job_info,
                  const mshn_data    *token_data,
                  const mshn_data    *user_cert,
                  const char         *passphrase,
                  mshn_data          *the_sig)
{
   int result = MSHN_OK;
   // Clear the output
   the_sig->the_length = 0;
   the_sig->the_data = NULL;

   // Now prepare the inputs for signing
   const int num_sign = 3;
   mshn_data work_array[num_sign];
   work_array[0].the_length = job_info->the_length;
   work_array[0].the_data   = job_info->the_data;
   work_array[1].the_length = token_data->the_length;
   work_array[1].the_data   = token_data->the_data;
   work_array[2].the_length = user_cert->the_length;
   work_array[2].the_data   = user_cert->the_data;

   mshn_data public_key;
   mshn_data private_key;
   result = msl_obj->mshn_sl_get_public_key(user_cert, &public_key);

   if (result != MSHN_OK) {
      err_out = msl_obj->mshn_sl_show_error(result);
      cout << "mshn_sl_get_public_key " << err_out << endl;
      mshn_free(err_out, NULL);
   }
   else{
      result = msl_obj->mshn_sl_get_private_key(
                     &public_key, &private_key);

      if (result != MSHN_OK) {
            err_out = msl_obj->mshn_sl_show_error(result);
            cout << "mshn_sl_get_private_key " << err_out << endl;
            mshn_free(err_out, NULL);
      }
   }
   if (result == MSHN_OK) {
      result = msl_obj->mshn_sl_sign(MSHN_ALG_DSA,
                                     MSHN_ALG_SHA,
                                    &private_key,
                                     passphrase,
                                     work_array,
                                     num_sign,
                                     the_sig);
```

```
      if (result != MSHN_OK) {
         err_out = msl_obj->mshn_sl_show_error(result);
         cout << "mshn_sl_sign " << err_out << endl;
         mshn_free(err_out, NULL);
      }
   }
   return result;
}

//==========================================================================
// decrypt data that came from the scheduler
int do_decrypt( mshn_data *resource_id,
                mshn_data *job_info,
                mshn_data *token_data,
                mshn_data *job_sess_key,
                mshn_data *core_cert,
                mshn_data *sched_sig)

{
   int result = MSHN_OK;

   // Now prepare the inputs for decryption
   const int num_signed_params = 5;
   const int num_params = 6;
   mshn_data work_array[num_params];
   work_array[0].the_length = resource_id->the_length;
   work_array[0].the_data   = resource_id->the_data;
   work_array[1].the_length = job_info->the_length;
   work_array[1].the_data   = job_info->the_data;
   work_array[2].the_length = token_data->the_length;
   work_array[2].the_data   = token_data->the_data;
   work_array[3].the_length = job_sess_key->the_length;
   work_array[3].the_data   = job_sess_key->the_data;
   work_array[4].the_length = core_cert->the_length;
   work_array[4].the_data   = core_cert->the_data;
   work_array[5].the_length = sched_sig->the_length;
   work_array[5].the_data   = sched_sig->the_data;

   mshn_data *dec_array[num_params];
   for (int i = 0; i < num_params; i++) {
      dec_array[i] = (mshn_data *)
                        mshn_malloc(sizeof(mshn_data *), NULL);
   }

   int numDecBuff, bytesDec;
   numDecBuff = num_params;

   result = msl_obj->mshn_sl_decrypt(&sym_key,
                                     MSHN_ALG_DES,
                                     work_array,
```

```
                                 num_params,
                                 dec_array,
                                 &numDecBuff,
                                 &bytesDec);

   if (result != MSHN_OK) {
      err_out = msl_obj->mshn_sl_show_error(result);
      cout << "mshn_sl_decrypt " << err_out << endl;
      mshn_free(err_out, NULL);
   }
   else {
   // replace input with decrypted copy

      resource_id->the_length  = dec_array[0]->the_length;
      resource_id->the_data    = dec_array[0]->the_data;
      job_info->the_length     = dec_array[1]->the_length;
      job_info->the_data       = dec_array[1]->the_data;
      token_data->the_length   = dec_array[2]->the_length;
      token_data->the_data     = dec_array[2]->the_data;
      job_sess_key->the_length = dec_array[3]->the_length;
      job_sess_key->the_data   = dec_array[3]->the_data;
      core_cert->the_length    = dec_array[4]->the_length;
      core_cert->the_data      = dec_array[4]->the_data;
      sched_sig->the_length    = dec_array[5]->the_length;
      sched_sig->the_data      = dec_array[5]->the_data;
   }

   return result;
}

//====================================================================
// decrypt data that came from the compute resource
int do_decrypt_resource( mshn_data *results,
                         mshn_data *app_signature,
                         mshn_data  job_sess_key)

{
   int result = MSHN_OK;

   // Now prepare the inputs for decryption
   const int num_params = 2;
   mshn_data work_array[num_params];
   work_array[0].the_length = results->the_length;
   work_array[0].the_data   = results->the_data;
   work_array[1].the_length = app_signature->the_length;
   work_array[1].the_data   = app_signature->the_data;

   mshn_data *dec_array[num_params];
   for (int i = 0; i < num_params; i++) {
      dec_array[i] = (mshn_data *)
                     mshn_malloc(sizeof(mshn_data *), NULL);
```

```
    }

    int numDecBuff, bytesDec;
    numDecBuff = num_params;

    result = msl_obj->mshn_sl_decrypt(&job_sess_key,
                                      MSHN_ALG_DES,
                                      work_array,
                                      num_params,
                                      dec_array,
                                      &numDecBuff,
                                      &bytesDec);

    if (result != MSHN_OK) {
       err_out = msl_obj->mshn_sl_show_error(result);
       cout << "mshn_sl_decrypt " << err_out << endl;
       mshn_free(err_out, NULL);
    }
    else {
    // replace input with decrypted copy
       results->the_length       = dec_array[0]->the_length;
       results->the_data         = dec_array[0]->the_data;
       app_signature->the_length = dec_array[1]->the_length;
       app_signature->the_data   = dec_array[1]->the_data;
    }

    return result;
}

//=====================================================================
// encrypt data that will be transmitted to the scheduler
int do_encrypt(
            mshn_data   *user_id,
            mshn_data   *cert,
            mshn_data   *sched_info,
            mshn_data   *signature)

{
    int result = MSHN_OK;

    // Now prepare the inputs for encryption
    const int num_params = 4;
    mshn_data work_array[num_params];
    work_array[0].the_length = user_id->the_length;
    work_array[0].the_data   = user_id->the_data;
    work_array[1].the_length = cert->the_length;
    work_array[1].the_data   = cert->the_data;
    work_array[2].the_length = sched_info->the_length;
    work_array[2].the_data   = sched_info->the_data;
    work_array[3].the_length = signature->the_length;
    work_array[3].the_data   = signature->the_data;
```

```
   mshn_data *enc_array[num_params];
   for (int i = 0; i < num_params; i++) {
      enc_array[i] = (mshn_data *)
                      mshn_malloc(sizeof(mshn_data *), NULL);
   }

   int numEncBuff, bytesEnc;
   numEncBuff = num_params;

   result = msl_obj->mshn_sl_encrypt(&sym_key,
                             MSHN_ALG_DES,
                             work_array,
                             num_params,
                             enc_array,
                             &numEncBuff,
                             &bytesEnc);

   if (result != MSHN_OK) {
      err_out = msl_obj->mshn_sl_show_error(result);
      cout << "mshn_sl_encrypt " << err_out << endl;
      mshn_free(err_out, NULL);
   }
   else {
      // replace input with encrypted copy
      user_id->the_length     = enc_array[0]->the_length;
      user_id->the_data       = enc_array[0]->the_data;
      cert->the_length        = enc_array[1]->the_length;
      cert->the_data          = enc_array[1]->the_data;
      sched_info->the_length  = enc_array[2]->the_length;
      sched_info->the_data    = enc_array[2]->the_data;
      signature->the_length   = enc_array[3]->the_length;
      signature->the_data     = enc_array[3]->the_data;
   }

   return result;
}

//====================================================================
// encrypt data going to the compute resource
int do_encrypt_resource(
            mshn_data   *job_info,
            mshn_data   *token_data,
            mshn_data   *user_cert,
            mshn_data   *signature)

{
   int result = MSHN_OK;

   // Now prepare the inputs for encryption
   const int num_params = 4;
```

```
   mshn_data work_array[num_params];
   work_array[0].the_length = job_info->the_length;
   work_array[0].the_data   = job_info->the_data;
   work_array[1].the_length = token_data->the_length;
   work_array[1].the_data   = token_data->the_data;
   work_array[2].the_length = user_cert->the_length;
   work_array[2].the_data   = user_cert->the_data;
   work_array[3].the_length = signature->the_length;
   work_array[3].the_data   = signature->the_data;

   mshn_data *enc_array[num_params];
   for (int i = 0; i < num_params; i++) {
      enc_array[i] = (mshn_data *)
                        mshn_malloc(sizeof(mshn_data *), NULL);
   }

   int numEncBuff, bytesEnc;
   numEncBuff = num_params;

   result = msl_obj->mshn_sl_encrypt(&sym_key,
                                     MSHN_ALG_DES,
                                     work_array,
                                     num_params,
                                     enc_array,
                                     &numEncBuff,
                                     &bytesEnc);

   if (result != MSHN_OK) {
      err_out = msl_obj->mshn_sl_show_error(result);
      cout << "mshn_sl_encrypt " << err_out << endl;
      mshn_free(err_out, NULL);
   }
   else {
       // replace input with encrypted copy
      job_info->the_length      = enc_array[0]->the_length;
      job_info->the_data        = enc_array[0]->the_data;
      token_data->the_length    = enc_array[1]->the_length;
      token_data->the_data      = enc_array[1]->the_data;
      user_cert->the_length     = enc_array[2]->the_length;
      user_cert->the_data       = enc_array[2]->the_data;
      signature->the_length     = enc_array[3]->the_length;
      signature->the_data       = enc_array[3]->the_data;
   }

   return result;
}
```

```
//======================================================================
// receive bundle from the scheduler,
// if necessary, decrypt and verify bundle
int do_recv_sched(const comm_security comm_sec,
                  mshn_data &resource_id,
                  mshn_data &job_info,
                  mshn_data &token_data,
                  mshn_data &job_sess_key,
                  mshn_data &core_cert,
                  mshn_data &sched_sig)
{
   int result = 0;
   int bytes_recv;

   result = recv_6_data(mc_obj,
                        &resource_id,
                        "Resource ID          ",
                        &job_info,
                        "Job Info             ",
                        &token_data,
                        "Security Token       ",
                        &job_sess_key,
                        "Job Session Key      ",
                        &core_cert,
                        "MSHN Core Certificate",
                        &sched_sig,
                        "Scheduler Signature  ",
                         bytes_recv);

   if (result == MSHN_COM_OK) {

      if ((comm_sec == COMSEC_CON) || (comm_sec == COMSEC_BOTH)) {
      // We must do decryption

      do_decrypt(&resource_id,
                 &job_info,
                 &token_data,
                 &job_sess_key,
                 &core_cert,
                 &sched_sig);
      }

      if ((comm_sec == COMSEC_INT) || (comm_sec == COMSEC_BOTH)) {
      // We must verify signature

         int sig_valid = 0;
         do_verify(resource_id,
                   job_info,
                   token_data,
                   job_sess_key,
                   core_cert,
```

```
                    sched_sig,
                    sig_valid);

         if (!sig_valid) {
            cout << "Signature failure.   Application terminated."
                 << endl;
            result == MSHN_INVALID_SIG;
         }
      }
   }

   return result;
}

//======================================================================
// receive results from compute resource,
// if necessary, decrypt and verify results
int do_recv_job(const comm_security comm_sec,
                  mshn_data &results,
                  mshn_data &app_signature,
                  mshn_data  job_sess_key)

{
   int result = 0;
   int bytes_recv;

   result = recv_2_data(mc_obj,
                        &results,
                        "Results                    ",
                        &app_signature,
                        "Application Signature",
                         bytes_recv);

   if (result == MSHN_COM_OK) {

      if ((comm_sec == COMSEC_CON) || (comm_sec == COMSEC_BOTH)) {
      // We must do decryption

            do_decrypt_resource(&results,
                                &app_signature,
                                 job_sess_key);
      }

      if ((comm_sec == COMSEC_INT) || (comm_sec == COMSEC_BOTH)) {
      // We must verify signature

         int sig_valid = 0;
         do_verify_resource(results,
                            app_signature,
                            job_sess_key,
                            sig_valid);
```

```
         if (!sig_valid) {
            cout << "Signature failure.   Application results are
                     NOT valid." << endl;
         }
      }
      cout << endl << "**********   APPLICATION   RESULTS   **********"
          << endl;
      cout << results.the_data << endl;

   }
   return result;
}

//==================================================================
// send job to compute resource and wait for results
// sign and encrypt job bundle if required
int do_send_job(const comm_security  comm_sec,
                      mshn_data resource_id,
                      mshn_data job_info,
                      mshn_data token_data,
                      mshn_data user_cert,
                const char      *passphrase)
{
   int result = 0;
   cout << "Sending job to compute resource..." << endl;
   // Set up connection to the Compute Resource

   result = mc_obj->mc_connect(resource_id.the_data,
                              PORT_CLIENT_RESOURCE);
   if (result == MSHN_COM_OK) {
      int bytes_sent;
      cout << "Connection made" << endl;
      mc_obj->mc_display(cout);

      // sign data before sending
      mshn_data signature;
      signature.the_data = NULL;
      signature.the_length = 0;

      if ((comm_sec == COMSEC_INT) || (comm_sec == COMSEC_BOTH)) {
            // We must do signature

            do_signature_resource(&job_info,
                                  &token_data,
                                  &user_cert,
                                   passphrase,
                                  &signature);
      }

      if ((comm_sec == COMSEC_CON) || (comm_sec == COMSEC_BOTH)) {
```

```
            // We must do encryption

            do_encrypt_resource(&job_info,
                                &token_data,
                                &user_cert,
                                &signature);
        }

        // send job input to the compute resource
        result = send_int_4_data(mc_obj,
                                comm_sec,
                                "Communications Security Option",
                               &job_info,
                                "Job Info                      ",
                               &token_data,
                                "Security Token                ",
                               &user_cert,
                                "User Certificate              ",
                               &signature,
                                "Client Signature              ",
                                bytes_sent);

            if (result == MSHN_COM_OK) {
               result = MSHN_OK;
               cout << "Sent (" << bytes_sent << ") bytes." << endl;

            }else{
               cout << "send_int_4_data: "
                    << mc_obj->mc_get_error(result)
                    << endl;
        }

   }else{
      cout << "mc_connect: " << mc_obj->mc_get_error(result) << endl;
   }

   return result;
}

//======================================================================
// prompt user for application to run
int do_choose_app(mshn_data &sched_info)
{
   int result = 0;
   sched_info.the_data = (unsigned char *)
                           mshn_malloc(BUFF_SIZE * 16, NULL);
   char choice[80];

   do {

      cout << endl;
```

```
        cout << "Choose an application to run:" << endl;
        cout << endl;
        cout << "    1) Application 1" << endl;
        cout << "    2) Application 2" << endl;
        cout << "    3) Application 3" << endl;
        cout << "    4) Application 4" << endl;
        cout << "    5) Application 5" << endl;

        cin.getline(choice, 80);
    }
    while (atoi(choice) < 1 || atoi(choice) > 5);

    switch (atoi(choice)) {

        case 1:
            strcpy(sched_info.the_data, "Application1");
            sched_info.the_length = strlen("Application1");
            break;

        case 2:
            strcpy(sched_info.the_data, "Application2");
            sched_info.the_length = strlen("Application2");
            break;

        case 3:
            strcpy(sched_info.the_data, "Application3");
            sched_info.the_length = strlen("Application3");
            break;

        case 4:
            strcpy(sched_info.the_data, "Application4");
            sched_info.the_length = strlen("Application4");
            break;

        case 5:
            strcpy(sched_info.the_data, "Application5");
            sched_info.the_length = strlen("Application5");
            break;

        default:
            break;
            // no default case
    }

    cout << "You chose application: " << sched_info.the_data << endl;
    return result;
}
```

```
//====================================================================
// send job request to the scheduler,
// if necessary, sign and encrypt the bundle
int do_send_request(const comm_security comm_sec,
                          mshn_data user_id,
                          mshn_data cert,
                          mshn_data sched_info,
                    const char *passphrase)
{
   int result = MSHN_OK;

   // Set up connection to the Scheduler
   result = mc_obj->mc_connect(IP_SCHEDULER, PORT_CLIENT_SCHEDULER);
   if (result == MSHN_COM_OK) {
      int bytes_sent;
      cout << "Connection made" << endl;
      mc_obj->mc_display(cout);

      // sign data before sending
      mshn_data signature;
      signature.the_data = NULL;
      signature.the_length = 0;

      if ((comm_sec == COMSEC_INT) || (comm_sec == COMSEC_BOTH)) {
            // We must do signature
            do_signature(&user_id,
                         &cert,
                         &sched_info,
                          passphrase,
                         &signature);
      }

      if ((comm_sec == COMSEC_CON) || (comm_sec == COMSEC_BOTH)) {
            // We must do encryption
            do_encrypt(&user_id,
                       &cert,
                       &sched_info,
                       &signature);
      }

      // send job input to the scheduler
      result = send_int_4_data(mc_obj,
                               comm_sec,
                               "Communications Security Option",
                               &user_id,
                               "User ID                       ",
                               &cert,
                               "User Certificate              ",
                               &sched_info,
                               "Schedule Info                 ",
                               &signature,
```

```
                                    "Client Signature              ",
                                     bytes_sent);

      if (result == MSHN_COM_OK) {
         result = MSHN_OK;
         cout << "Sent (" << bytes_sent << ") bytes." << endl;

      }else{
         cout << "send_int_4_data: "
              << mc_obj->mc_get_error(result)
              << endl;
      }

   }else{
      cout << "mc_connect: " << mc_obj->mc_get_error(result) << endl;
   }

   return result;
}


//===================================================================
int main(int, char *[]) {

   comm_security com_sec_option;
   cert_checking cert_valid_level;
   char *passphrase;

   mshn_data user_cert;
   mshn_data user_id;
   mshn_data sched_info;

   mshn_data resource_id;
   mshn_data job_info;
   mshn_data token_data;
   mshn_data job_sess_key;
   mshn_data core_cert;
   mshn_data sched_sig;

   mshn_data results;
   mshn_data app_signature;

   mc_obj = new mshn_com();

#if defined (DEBUG_MSHN_COM)
   cout << "Hello World" << endl;
   mc_obj->mc_display(cout);
#endif
   char *err_out;
   char dummy_in[80];
```

```
msl_obj = new mshn_sl();
int result = msl_obj->mshn_sl_init("dummy file");
if (result != MSHN_OK) {
   err_out = msl_obj->mshn_sl_show_error(result);
   cout << "mshn_sl_init " << err_out << endl;
   mshn_free(err_out, NULL);
}else{
   // get the shared symmetric key
   read_data(&sym_key, key_fname, BUFF_SIZE);

   clrscr();
   cout << endl << "*******************  MSHN CLIENT SHELL  ";
   cout         << "*******************" << endl;
   cout << endl;
   cout << "This program will allow you to submit a job to MSHN."
        << endl;
   cout << "You must identify yourself, (certificate name and
            passphrase)" << endl;
   cout << "and select the application you wish MSHN to execute."
        << endl;
   cout << endl;
   cout << "Press enter to continue." << endl;
   cin.getline(dummy_in, 80);

   do_register(msl_obj,
               user_id,
               user_cert,
              &passphrase,
               com_sec_option,
               cert_valid_level);

   char quit[80];

   do {

      cout << endl;
      cout << "Choose an application to submit to MSHN." << endl;
      cout << "Submit job, wait for results, display results."
           << endl;
      cout << endl;
      cout << "Continue? Enter y/n. ";
      cin.getline(quit, 80);

      if (quit[0] == 'y' || quit[0] == 'Y') {
         do_choose_app(sched_info);

         // send job request to the scheduler
         do_send_request(com_sec_option,
                         user_id,
                         user_cert,
                         sched_info,
```

```
                                  passphrase);

            // receive response from scheduler
            do_recv_sched(com_sec_option,
                          resource_id,
                          job_info,
                          token_data,
                          job_sess_key,
                          core_cert,
                          sched_sig);

            // make sure the connection to scheduler is closed
            mc_obj->mc_close();

            // send job to compute resource and wait for result
            do_send_job(com_sec_option,
                        resource_id,
                        job_info,
                        token_data,
                        user_cert,
                        passphrase);

            do_recv_job(com_sec_option,
                        results,
                        app_signature,
                        job_sess_key);

            // make sure the connection to compute resource is closed
            mc_obj->mc_close();
        }
    } while (!(quit[0] == 'n' || quit[0] == 'N'));
  }
  return 0;
}
```

B. SCHEDULER.CPP

```
//*******************************************************
// File:  scheduler.cpp
// Name:  David Shifflett
//
// Project: MSHN
//
// Operating Environment: Windows 95/Windows NT
// Compiler: Borland C++ for Windows
// Date: 18 MAY 98
//
// Description: MSHN demonstration scheduler process
//*******************************************************
```

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>              // for randomize()
#include <stdio.h>
#include <fstream.h>
#include "mshn_sl.h"
#include "mshn_mem.h"
#include "mshn_com.h"
#include "mshn_defs.h"
#include "mshn_err.h"
#include "mshn_types.h"
#include "mshn_demo.h"
#include "commutil.h"
#include "showutil.h"

char *err_out;
mshn_sl   *msl_obj;
mshn_com *mc_obj;
mshn_data sym_key;
const int BUFF_SIZE = 1024;
cert_checking core_cert_check;
comm_security core_comm_sec;
char *passphrase;
mshn_data core_cert, core_id;
char dummy_in[80];

//======================================================================
// Sign the data to be transmitted
int   do_encrypt(
                  comm_security     c_sec,
                  mshn_data         *resource_id,
                  mshn_data         *job_info,
                  mshn_data         *the_token,
                  mshn_data         *job_sess_key,
                  mshn_data         *core_cert,
                  mshn_data         *the_sig)
{
   int result = MSHN_OK;

   if ((c_sec == COMSEC_CON) || (c_sec == COMSEC_BOTH)) {
      // We must do encryption
      // Now prepare the inputs for signing
      const int num_params = 6;
      mshn_data work_array[num_params];
      work_array[0].the_length = resource_id->the_length;
      work_array[0].the_data   = resource_id->the_data;
      work_array[1].the_length = job_info->the_length;
      work_array[1].the_data   = job_info->the_data;
      work_array[2].the_length = the_token->the_length;
      work_array[2].the_data   = the_token->the_data;
      work_array[3].the_length = job_sess_key->the_length;
```

```
        work_array[3].the_data   = job_sess_key->the_data;
        work_array[4].the_length = core_cert->the_length;
        work_array[4].the_data   = core_cert->the_data;
        work_array[5].the_length = the_sig->the_length;
        work_array[5].the_data   = the_sig->the_data;

        mshn_data *enc_array[num_params];
        for (int i = 0; i < num_params; i++) {
           enc_array[i] = (mshn_data *)
                          mshn_malloc(sizeof(mshn_data *), NULL);
        }

        int numEncBuff, bytesEnc;
        numEncBuff = num_params;

        result = msl_obj->mshn_sl_encrypt(&sym_key,
                                          MSHN_ALG_DES,
                                          work_array,
                                          num_params,
                                          enc_array,
                                          &numEncBuff,
                                          &bytesEnc);

        if (result != MSHN_OK) {
           err_out = msl_obj->mshn_sl_show_error(result);
           cout << "mshn_sl_sign " << err_out << endl;
           mshn_free(err_out, NULL);
        }else{
           // copy the decrypted buffers to the output buffers
           resource_id->the_length     = enc_array[0]->the_length;
           resource_id->the_data       = enc_array[0]->the_data;
           job_info->the_length        = enc_array[1]->the_length;
           job_info->the_data          = enc_array[1]->the_data;
           the_token->the_length       = enc_array[2]->the_length;
           the_token->the_data         = enc_array[2]->the_data;
           job_sess_key->the_length    = enc_array[3]->the_length;
           job_sess_key->the_data      = enc_array[3]->the_data;
           core_cert->the_length       = enc_array[4]->the_length;
           core_cert->the_data         = enc_array[4]->the_data;
           the_sig->the_length         = enc_array[5]->the_length;
           the_sig->the_data           = enc_array[5]->the_data;
        }
    }
    return result;
}

//==================================================================
// Sign the data to be transmitted
int   do_signature(
                   comm_security      c_sec,
                   const mshn_data    *resource_id,
```

```
                        const mshn_data    *job_info,
                        const mshn_data    *the_token,
                        const mshn_data    *job_sess_key,
                        const mshn_data    *core_cert,
                        mshn_data          *the_sig)
{
   int result = MSHN_OK;
   // Clear the output
   the_sig->the_length = 0;
   the_sig->the_data = NULL;

   if ((c_sec == COMSEC_INT) || (c_sec == COMSEC_BOTH)) {
      // We must do signature

      // Now prepare the inputs for signing
      const int num_sign = 5;
      mshn_data work_array[num_sign];
      work_array[0].the_length = resource_id->the_length;
      work_array[0].the_data   = resource_id->the_data;
      work_array[1].the_length = job_info->the_length;
      work_array[1].the_data   = job_info->the_data;
      work_array[2].the_length = the_token->the_length;
      work_array[2].the_data   = the_token->the_data;
      work_array[3].the_length = job_sess_key->the_length;
      work_array[3].the_data   = job_sess_key->the_data;
      work_array[4].the_length = core_cert->the_length;
      work_array[4].the_data   = core_cert->the_data;

      mshn_data public_key;
      mshn_data private_key;
      result = msl_obj->mshn_sl_get_public_key(core_cert, &public_key);

      if (result != MSHN_OK) {
            err_out = msl_obj->mshn_sl_show_error(result);
            cout << "mshn_sl_get_public_key " << err_out << endl;
            mshn_free(err_out, NULL);
      }else{
            result = msl_obj->mshn_sl_get_private_key(
                                 &public_key, &private_key);

            if (result != MSHN_OK) {
                  err_out = msl_obj->mshn_sl_show_error(result);
                  cout << "mshn_sl_get_private_key " << err_out << endl;
                  mshn_free(err_out, NULL);
            }
      }

      if (result == MSHN_OK) {
            result = msl_obj->mshn_sl_sign(MSHN_ALG_DSA,
                                 MSHN_ALG_SHA,
                                 &private_key,
```

```
                                          passphrase,
                                          work_array,
                                          num_sign,
                                          the_sig);

                if (result != MSHN_OK) {
                      err_out = msl_obj->mshn_sl_show_error(result);
                      cout << "mshn_sl_sign " << err_out << endl;
                      mshn_free(err_out, NULL);
                }
          }
          // Now clean up the allocated memory
          mshn_free(public_key.the_data, NULL);
          mshn_free(private_key.the_data, NULL);
    }
    return result;
}

//=====================================================================
// Process the received data
//
// this routine will do the following
//
// create unique request_id.
// consult scheduler
// create unique job session key
// write user_id, security level, request id, session key to REQUEST DB
// Fill in security token (output), encrypt session key with resource's
// public key
// Encrypt session key (output) with users's public key
// Send to client, resource id (from scheduler), job info (from
// scheduler), security token, encrypted session key, MSHN core
// certificate, signature
//
int   do_schedule(
                  mshn_com            *new_conn,
                  comm_security       c_sec,
                  mshn_data           user_id,
                  mshn_data           blob)
{
    int result = MSHN_OK;
    // Lets display what we received
    cout << endl << endl << endl;
    show_data_char_ptr((CSSM_DATA_PTR)&user_id, "User Id", 1);
    show_data_char_ptr((CSSM_DATA_PTR)&blob, "Job Info", 1);
    cout << endl << "Press enter "; cin.getline(dummy_in, 80);

    // create unique request_id.
    int request_id = random(RAND_MAX);
    mshn_data resource_id, job_info;
    mshn_data job_sess_key, the_salt;
```

```
mshn_data the_sig;
mshn_token the_token;
mshn_data token_data;

// consult scheduler
// FOR NOW JUST FAKE IT
resource_id.the_data = strdup(IP_RESOURCE_1);
resource_id.the_length = strlen(IP_RESOURCE_1) + 1;
job_info.the_data = strdup("Run the job as fast as you can.");
job_info.the_length = strlen(job_info.the_data) + 1;

// create unique job session key
char key_label[] = "DES Key Label";
result = msl_obj->mshn_sl_sym_key_gen(
                     MSHN_ALG_DES,
                     40,
                     the_salt,
                     key_label,
                    &job_sess_key);

if (result != MSHN_OK) {
   err_out = msl_obj->mshn_sl_show_error(result);
   cout << "mshn_sl_sym_key_gen " << err_out << endl;
   mshn_free(err_out, NULL);
}else{
   // write user_id, security level, request id, session key
   // to REQUEST DB
   result = add_to_req_db(reqdb_fname, &user_id,
                          c_sec, request_id, &job_sess_key);
}

if (result == MSHN_OK) {
   // Fill in security token (output), encrypt session key
   // with resource's public key
   // WE HAVE NO PUBLIC/PRIVATE KEY ENCRYPT/DECRYPT, SO SEND
   // THE KEY IN THE CLEAR
   the_token.request_id = request_id;
   the_token.job_session_key.the_length = job_sess_key.the_length;
   the_token.job_session_key.the_data   = job_sess_key.the_data;
   // convert the token to a mshn_data
   token_to_mshn_data(the_token, &token_data);

   // Encrypt session key (output) with users's public key
   // WE HAVE NO PUBLIC/PRIVATE KEY ENCRYPT/DECRYPT, SO SEND
   // THE KEY IN THE CLEAR

   // Sign the data to be transmitted
   result = do_signature(c_sec, &resource_id, &job_info,
            &token_data, &job_sess_key, &core_cert, &the_sig);
}
```

```
    mshn_data temp_cert;
    temp_cert.the_data = NULL;
    if (result == MSHN_OK) {
       // do an encrypt if necessary
       temp_cert.the_length = core_cert.the_length;
       temp_cert.the_data = (unsigned char *)
                              mshn_malloc(temp_cert.the_length, NULL);
       memcpy(temp_cert.the_data, core_cert.the_data,
              temp_cert.the_length);
       result = do_encrypt(c_sec, &resource_id, &job_info,
                &token_data, &job_sess_key, &temp_cert, &the_sig);
    }

    if (result == MSHN_OK) {
       // Send to client, resource id (from scheduler), job info
       // (from scheduler), security token, encrypted session key,
       // MSHN core certificate, signature
       int bytes_sent;
       result = send_6_data(new_conn,
                            &resource_id,  "Resource ID            ",
                            &job_info,     "Job information        ",
                            &token_data,   "Security token         ",
                            &job_sess_key, "Job session key        ",
                            &temp_cert,    "MSHN core certificate",
                            &the_sig,      "Scheduler signature  ",
                            bytes_sent);
       if (result == MSHN_COM_OK) {
            result = MSHN_OK;
       }
    }

    // Free up allocated memory
    mshn_free(temp_cert.the_data, NULL);
    mshn_free(resource_id.the_data, NULL);
    mshn_free(job_info.the_data, NULL);
    mshn_free(job_sess_key.the_data, NULL);
    mshn_free(the_sig.the_data, NULL);
    mshn_free(token_data.the_data, NULL);
    return result;
}

//=====================================================================
// Verify the received data
int   verify_request(
                 mshn_com     *new_conn,
                 int           comm_sec,
                 mshn_data     user_id,
                 mshn_data     cert,
                 mshn_data     blob,
                 mshn_data     signature)
{
```

```
int result = MSHN_OK;
const int num_params = 4;
const int num_signed_params = 3;

// Prepare the inputs for encryption/decryption/signature
// verification
mshn_data work_array[num_params];
work_array[0].the_length = user_id.the_length;
work_array[0].the_data   = user_id.the_data;
work_array[1].the_length = cert.the_length;
work_array[1].the_data   = cert.the_data;
work_array[2].the_length = blob.the_length;
work_array[2].the_data   = blob.the_data;
work_array[3].the_length = signature.the_length;
work_array[3].the_data   = signature.the_data;

mshn_data *dec_array[num_params];
for (int idx=0; idx<num_params; idx++) {
   dec_array[idx] = (mshn_data *)
                    mshn_malloc(sizeof(mshn_data *), NULL);
}

// Determine how to handle the inputs
// What decryption/signature verification is necessary?
   comm_security c_sec = comm_sec;
if ((c_sec == COMSEC_CON) || (c_sec == COMSEC_BOTH)) {
   // We must do decryption
   int numdecBuff, bytesdec;
   numdecBuff = num_params;

   result = msl_obj->mshn_sl_decrypt(&sym_key, MSHN_ALG_DES,
            work_array, num_params, dec_array, &numdecBuff,
            &bytesdec);
   if (result != MSHN_OK) {
         err_out = msl_obj->mshn_sl_show_error(result);
         cout << "mshn_sl_decrypt " << err_out << endl;
         mshn_free(err_out, NULL);
   }else{
         // Prepare for the next steps, by putting the decrypted
         // buffers into work_array
         for (int idx=0; idx<num_params; idx++) {
               work_array[idx].the_length =
                     dec_array[idx]->the_length;
               work_array[idx].the_data = dec_array[idx]->the_data;
         }
   }
}

if (result == MSHN_OK) {
   if ((c_sec == COMSEC_INT) || (c_sec == COMSEC_BOTH)) {
      // We must do signature verification
```

```
      // First verify the certificate
      mshn_data *certificate = &work_array[1];
      result = do_cert_check(msl_obj, core_cert_check,
                             certificate);

      // Now get the public key
      mshn_data public_key;
      if (result == MSHN_OK) {
         result = msl_obj->mshn_sl_get_public_key(
                           certificate, &public_key);

         if (result != MSHN_OK) {
               err_out = msl_obj->mshn_sl_show_error(result);
               cout << "mshn_sl_get_public_key " << err_out << endl;
               mshn_free(err_out, NULL);
         }
      }
      // Now do signature verification
      if (result == MSHN_OK) {
         int sig_valid = 0;
         result = msl_obj->mshn_sl_sig_verify(MSHN_ALG_DSA,
                           MSHN_ALG_SHA,
                           &public_key,
                           work_array,
                           num_signed_params,
                           &work_array[3], // this one is the sig
                           &sig_valid);

         if (result != MSHN_OK) {
               err_out = msl_obj->mshn_sl_show_error(result);
               cout << "mshn_sl_sig_verify " << err_out << endl;
               mshn_free(err_out, NULL);
         }else{
               if (sig_valid == MSHN_FALSE) {
               result = MSHN_INVALID_SIG;
               err_out = msl_obj->mshn_sl_show_error(result);
               cout << "mshn_sl_sig_verify " << err_out << endl;
               mshn_free(err_out, NULL);
         }
      }
   }
}

if (result == MSHN_OK) {
   // If we get here, the inputs have been decrypted and verified
   // so lets do something with them
   result = do_schedule(new_conn, c_sec,
                        work_array[0],       // this is the user_id
                        work_array[2]);      // this is the blob
}
```

```
      for (int idx=0; idx<num_params; idx++) {
      mshn_free(dec_array[idx], NULL);
   }
   return result;
}

//=====================================================================
int handle_request()
{
      int result = MSHN_OK;

   // Now accept connections from the client/laptop
   mshn_com    *new_conn;
   result = mc_obj->mc_accept("Client",
                              PORT_CLIENT_SCHEDULER, &new_conn);
   if (result == MSHN_COM_OK) {
      result = MSHN_OK;
#if defined (DEBUG_MSHN_COM)
      cout << "Accepted connection" << endl;
      new_conn->mc_display(cout);
#endif

      int bytes_rec, comm_sec;
      mshn_data user_id, cert, blob, signature;

      // get the input from the client/laptop
      result = recv_int_4_data(new_conn,
                         &comm_sec,  "Communication security option",
                         &user_id,   "Client ID                    ",
                         &cert,      "Client certificate           ",
                         &blob,      "Job information              ",
                         &signature, "Client signature             ",
                          bytes_rec);
      if (result == MSHN_COM_OK) {
         result = MSHN_OK;
         cout << "Received (" << bytes_rec << ") bytes." << endl;

         // Now process the received data
         result = verify_request(new_conn, comm_sec, user_id,
                   cert, blob, signature);
      }else{
            cout << "recv_int_4_data: "
                 << new_conn->mc_get_error(result) << endl;
      }
      // make sure the connection is closed
      new_conn->mc_close();
   }else{
      cout << "mc_accept: " << mc_obj->mc_get_error(result) << endl;
   }
   // make sure the connection is closed
   mc_obj->mc_close();
```

```
    return result;
}

//=====================================================================
int main(int, char *[]) {
    randomize();            // init random number generator
    mc_obj = new mshn_com();
    msl_obj = new mshn_sl();
    core_cert_check = CERT_NONE;
    int result = msl_obj->mshn_sl_init("dummy file");
    if (result != MSHN_OK) {
        err_out = msl_obj->mshn_sl_show_error(result);
        cout << "mshn_sl_init " << err_out << endl;
        mshn_free(err_out, NULL);
    }else{
        // Get the shared symmetric key for encryption/decryption
        read_data(&sym_key, key_fname, BUFF_SIZE);


        do_register(msl_obj, core_id, core_cert, &passphrase,
                              core_comm_sec, core_cert_check);

        // now receive and handle incoming scheduling requests
        while (result == MSHN_OK) {
              result = handle_request();
        }
    }
    return 0;
}
```

C. RESOURCE STATUS SERVER

```
//*******************************************************
// File:   rss.cpp
// Name:   David Shifflett
//
// Project: MSHN
//
// Operating Environment: Windows 95/Windows NT
// Compiler: Borland C++ for Windows
// Date:
//
// Description: MSHN demonstration Resource Status Server process
//*******************************************************

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
```

```
#include "mshn_sl.h"
#include "mshn_mem.h"
#include "mshn_com.h"
#include "mshn_defs.h"
#include "mshn_err.h"
#include "mshn_types.h"
#include "mshn_demo.h"
#include "commutil.h"
#include "showutil.h"

char *err_out;
mshn_sl  *msl_obj;
mshn_com *mc_obj;
mshn_data sym_key;
const int BUFF_SIZE = 1024;
cert_checking core_cert_check;
comm_security core_comm_sec;
char *passphrase;
mshn_data core_cert, core_id;
char dummy_in[80];

//=======================================================================
// Verify the received data
int   verify_request(
                  int           request_id,
                  mshn_data     resource_id,
                  mshn_data     stat_info,
                  mshn_data     signature)
{
   int result = MSHN_OK;
   const int num_params = 4;
   const int num_encrypted_params = 3;
   const int num_signed_params = 3;
   char dummy[80];

   // Prepare the inputs for encryption/decryption/signature
   // verification
   mshn_data work_array[num_params];

   mshn_data *dec_array[num_params];
   for (int idx=0; idx<num_params; idx++) {
      dec_array[idx] = (mshn_data *)
                     mshn_malloc(sizeof(mshn_data *), NULL);
   }

   // We need to use the request_id to look up the data in the
   // REQUEST DB
   mshn_data user_id, sess_key;
   int comm_sec;
   comm_security c_sec;
   result = get_from_req_db(reqdb_fname, request_id, &user_id,
```

```
            comm_sec, &sess_key);
      if (result != MSHN_OK) {
         err_out = msl_obj->mshn_sl_show_error(result);
         cout << "get_from_req_db " << err_out << endl;
         mshn_free(err_out, NULL);
      }else{
         cout << endl << "For request ID: " << request_id << endl;
         cout << "FOUND: communication security: " << comm_sec << endl;
         memcpy(dummy, user_id.the_data, user_id.the_length);
         dummy[user_id.the_length] = '\0';
         cout << "FOUND: user id                : " << dummy << endl;
         cout << "FOUND: session key: " << endl;
         show_pointer (sess_key.the_data, sess_key.the_length,
                       "session key");

         // Determine how to handle the inputs
         // What decryption/signature verification is necessary?
         c_sec = comm_sec;
         if ((c_sec == COMSEC_CON) || (c_sec == COMSEC_BOTH)) {
            // We must do decryption
            int numdecBuff, bytesdec;
            numdecBuff = num_encrypted_params;
            work_array[0].the_length = resource_id.the_length;
            work_array[0].the_data   = resource_id.the_data;
            work_array[1].the_length = stat_info.the_length;
            work_array[1].the_data   = stat_info.the_data;
            work_array[2].the_length = signature.the_length;
            work_array[2].the_data   = signature.the_data;

            result = msl_obj->mshn_sl_decrypt(&sess_key, MSHN_ALG_DES,
                     work_array, num_encrypted_params, dec_array,
                     &numdecBuff, &bytesdec);
            if (result != MSHN_OK) {
               err_out = msl_obj->mshn_sl_show_error(result);
               cout << "mshn_sl_decrypt " << err_out << endl;
               mshn_free(err_out, NULL);
            }else{
               // Prepare for the next steps, by putting the decrypted
               // buffers into work_array
               work_array[0].the_length = sizeof(int);
               // This field can't be encrypted
               work_array[0].the_data   = (unsigned char *)&request_id;
               for (int idx=0; idx<num_encrypted_params; idx++) {
                   work_array[idx + 1].the_length =
                     dec_array[idx]->the_length;
                   work_array[idx + 1].the_data = dec_array[idx]->the_data;
               }
            }
         }else{
               work_array[0].the_length = sizeof(int);
               // This field can't be encrypted
```

```
            work_array[0].the_data   = (unsigned char *)&request_id;
            work_array[1].the_length = resource_id.the_length;
            work_array[1].the_data   = resource_id.the_data;
            work_array[2].the_length = stat_info.the_length;
            work_array[2].the_data   = stat_info.the_data;
            work_array[3].the_length = signature.the_length;
            work_array[3].the_data   = signature.the_data;
        }
    }
    if (result == MSHN_OK) {
        if ((c_sec == COMSEC_INT) || (c_sec == COMSEC_BOTH)) {
            // We must do signature verification
            int sig_valid = 0;
            result = msl_obj->mshn_sl_sig_verify(MSHN_ALG_DES,
                            MSHN_ALG_MD5,
                            &sess_key,
                            work_array,
                            num_signed_params,
                            &work_array[3], // this one is the signature
                            &sig_valid);

            if (result != MSHN_OK) {
                err_out = msl_obj->mshn_sl_show_error(result);
                cout << "mshn_sl_sig_verify " << err_out << endl;
                mshn_free(err_out, NULL);
            }else{
                if (sig_valid == MSHN_FALSE) {
                    result = MSHN_INVALID_SIG;
                    err_out = msl_obj->mshn_sl_show_error(result);
                    cout << "mshn_sl_sig_verify " << err_out << endl;
                    mshn_free(err_out, NULL);
                }
            }
        }
    }

    if (result == MSHN_OK) {
        // If we get here, the inputs have been decrypted and verified
        // so lets do something with them
        cout << "Received update from resource" << endl;
        memcpy(dummy, work_array[1].the_data, work_array[1].the_length);
        dummy[work_array[1].the_length] = '\0';
        cout << "Resource id                 : " << dummy << endl;
        memcpy(dummy, work_array[2].the_data, work_array[2].the_length);
        dummy[work_array[2].the_length] = '\0';
        cout << "Resource status info        : " << dummy << endl;
    }

        for (int idx=0; idx<num_params; idx++) {
        mshn_free(dec_array[idx], NULL);
    }
```

```
   return result;
}

//==========================================================================
int handle_request()
{
   int result = MSHN_OK;

   // Now accept connections from the client/laptop
   mshn_com   *new_conn;
   result = mc_obj->mc_accept("Resource", PORT_RESOURCE_RSS, &new_conn);
   if (result == MSHN_COM_OK) {
      result = MSHN_OK;
#if defined (DEBUG_MSHN_COM)
      cout << "Accepted connection" << endl;
      new_conn->mc_display(cout);
#endif

      int bytes_rec, req_id;
      mshn_data resource_id, stat_info, signature;

      // get the input from the client/laptop
      result = recv_int_3_data(new_conn,
                               &req_id,        "Job request ID       ",
                               &resource_id,   "Resource ID          ",
                               &stat_info,     "Resource status info",
                               &signature,     "Resource signature   ",
                               bytes_rec);
      if (result == MSHN_COM_OK) {
         result = MSHN_OK;
         cout << "Received (" << bytes_rec << ") bytes." << endl;

         // Now process the received data
         result = verify_request(req_id, resource_id, stat_info,
                     signature);
      }else{
         cout << "recv_int_3_data: " << new_conn->mc_get_error(result)
              << endl;
      }
      // make sure the connection is closed
      new_conn->mc_close();
   }else{
      cout << "mc_accept: " << mc_obj->mc_get_error(result) << endl;
   }
   // make sure the connection is closed
   mc_obj->mc_close();
   return result;
}
```

```
//======================================================================
int main(int, char *[]) {
   mc_obj = new mshn_com();
   msl_obj = new mshn_sl();
   core_cert_check = CERT_NONE;
   int result = msl_obj->mshn_sl_init("dummy file");
   if (result != MSHN_OK) {
      err_out = msl_obj->mshn_sl_show_error(result);
      cout << "mshn_sl_init " << err_out << endl;
      mshn_free(err_out, NULL);
   }else{
      // Get the shared symmetric key for encryption/decryption
      read_data(&sym_key, key_fname, BUFF_SIZE);

      do_register(msl_obj, core_id, core_cert, &passphrase,
                        core_comm_sec, core_cert_check);

      // now receive and handle incoming scheduling requests
      while (result == MSHN_OK) {
            result = handle_request();
      }
   }
   return 0;
}
```

D. RESOURCE REQUIREMENTS DATABASE

```
//*******************************************************
// File:  rrd.cpp
// Name:  David Shifflett
//
// Project: MSHN
//
// Operating Environment: Windows 95/Windows NT
// Compiler: Borland C++ for Windows
// Date:
//
// Description: MSHN demonstration Resource Requirements
// Database process
//
//*******************************************************

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include "mshn_sl.h"
#include "mshn_mem.h"
#include "mshn_com.h"
#include "mshn_defs.h"
```

```
#include "mshn_err.h"
#include "mshn_types.h"
#include "mshn_demo.h"
#include "commutil.h"
#include "showutil.h"

char *err_out;
mshn_sl  *msl_obj;
mshn_com *mc_obj;
mshn_data sym_key;
const int BUFF_SIZE = 1024;
cert_checking core_cert_check;
comm_security core_comm_sec;
char *passphrase;
mshn_data core_cert, core_id;
char dummy_in[80];

//====================================================================
// Verify the received data
int   verify_request(
                  int          request_id,
                  mshn_data    resource_id,
                  mshn_data    req_info,
                  mshn_data    signature)
{
   int result = MSHN_OK;
   const int num_params = 4;
   const int num_encrypted_params = 3;
   const int num_signed_params = 3;
   char dummy[80];

   // Prepare the inputs for encryption/decryption/signature
   // verification
   mshn_data work_array[num_params];

   mshn_data *dec_array[num_params];
   for (int idx=0; idx<num_params; idx++) {
      dec_array[idx] = (mshn_data *)
                        mshn_malloc(sizeof(mshn_data *), NULL);
   }

   // We need to use the request_id to look up the data in the
   // REQUEST DB
   mshn_data user_id, sess_key;
   int comm_sec;
   comm_security c_sec;
   result = get_from_req_db(reqdb_fname, request_id, &user_id,
            comm_sec, &sess_key);
   if (result != MSHN_OK) {
      err_out = msl_obj->mshn_sl_show_error(result);
      cout << "get_from_req_db " << err_out << endl;
```

```
    mshn_free(err_out, NULL);
}else{
    cout << endl << "For request ID: " << request_id << endl;
    cout << "FOUND: communication security: " << comm_sec << endl;
    memcpy(dummy, user_id.the_data, user_id.the_length);
    dummy[user_id.the_length] = '\0';
    cout << "FOUND: user id                 : " << dummy << endl;
    cout << "FOUND: session key: " << endl;
    show_pointer (sess_key.the_data, sess_key.the_length,
                  "session key");

    // Determine how to handle the inputs
    // What decryption/signature verification is necessary?
    c_sec = comm_sec;
    if ((c_sec == COMSEC_CON) || (c_sec == COMSEC_BOTH)) {
       // We must do decryption
       int numdecBuff, bytesdec;
       numdecBuff = num_encrypted_params;
       work_array[0].the_length = resource_id.the_length;
       work_array[0].the_data   = resource_id.the_data;
       work_array[1].the_length = req_info.the_length;
       work_array[1].the_data   = req_info.the_data;
       work_array[2].the_length = signature.the_length;
       work_array[2].the_data   = signature.the_data;


       result = msl_obj->mshn_sl_decrypt(&sess_key, MSHN_ALG_DES,
                          work_array, num_encrypted_params, dec_array,
                          &numdecBuff, &bytesdec);
       if (result != MSHN_OK) {
          err_out = msl_obj->mshn_sl_show_error(result);
          cout << "mshn_sl_decrypt " << err_out << endl;
          mshn_free(err_out, NULL);
       }else{
          // Prepare for the next steps, by putting the decrypted
          // buffers into work_array
          work_array[0].the_length = sizeof(int);
          // This field can't be encrypted
          work_array[0].the_data   = (unsigned char *)&request_id;
          for (int idx=0; idx<num_encrypted_params; idx++) {
             work_array[idx + 1].the_length =
                dec_array[idx]->the_length;
             work_array[idx + 1].the_data = dec_array[idx]->the_data;
          }
       }
    }else{
          work_array[0].the_length = sizeof(int);
          // This field can't be encrypted
          work_array[0].the_data   = (unsigned char *)&request_id;
          work_array[1].the_length = resource_id.the_length;
          work_array[1].the_data   = resource_id.the_data;
```

```
            work_array[2].the_length = req_info.the_length;
            work_array[2].the_data   = req_info.the_data;
            work_array[3].the_length = signature.the_length;
            work_array[3].the_data   = signature.the_data;
      }
   }

   if (result == MSHN_OK) {
      if ((c_sec == COMSEC_INT) || (c_sec == COMSEC_BOTH)) {
         // We must do signature verification
         int sig_valid = 0;
         result = msl_obj->mshn_sl_sig_verify(MSHN_ALG_DES,
                        MSHN_ALG_MD5,
                        &sess_key,
                        work_array,
                        num_signed_params,
                        &work_array[3], // this one is the signature
                        &sig_valid);

         if (result != MSHN_OK) {
            err_out = msl_obj->mshn_sl_show_error(result);
            cout << "mshn_sl_sig_verify " << err_out << endl;
            mshn_free(err_out, NULL);
         }else{
            if (sig_valid == MSHN_FALSE) {
               result = MSHN_INVALID_SIG;
               err_out = msl_obj->mshn_sl_show_error(result);
               cout << "mshn_sl_sig_verify " << err_out << endl;
               mshn_free(err_out, NULL);
            }
         }
      }
   }

   if (result == MSHN_OK) {
      // If we get here, the inputs have been decrypted and verified
      // so lets do something with them
      cout << "Received update from resource" << endl;
      memcpy(dummy, work_array[1].the_data, work_array[1].the_length);
      dummy[work_array[1].the_length] = '\0';
      cout << "Resource id                : " << dummy << endl;
      memcpy(dummy, work_array[2].the_data, work_array[2].the_length);
      dummy[work_array[2].the_length] = '\0';
      cout << "Requirements info          : " << dummy << endl;
   }

      for (int idx=0; idx<num_params; idx++) {
      mshn_free(dec_array[idx], NULL);
   }
   return result;
}
```

```
//=====================================================================
int handle_request()
{
   int result = MSHN_OK;

   // Now accept connections from the client/laptop
   mshn_com   *new_conn;
   result = mc_obj->mc_accept("Resource", PORT_RESOURCE_RRD, &new_conn);
   if (result == MSHN_COM_OK) {
      result = MSHN_OK;
#if defined (DEBUG_MSHN_COM)
      cout << "Accepted connection" << endl;
      new_conn->mc_display(cout);
#endif

   int bytes_rec, req_id;
   mshn_data resource_id, req_info, signature;

   // get the input from the client/laptop
   result = recv_int_3_data(new_conn,
                     &req_id,        "Job request ID               ",
                     &resource_id,   "Resource ID                  ",
                     &req_info,      "Resource requirements info",
                     &signature,     "Resource signature           ",
                      bytes_rec);

   if (result == MSHN_COM_OK) {
      result = MSHN_OK;
      cout << "Received (" << bytes_rec << ") bytes." << endl;

      // Now process the received data
      result = verify_request(req_id, resource_id, req_info, signature);
   }else{
      cout << "recv_int_3_data: " << new_conn->mc_get_error(result)
           << endl;
   }
   // make sure the connection is closed
   new_conn->mc_close();
   }else{
      cout << "mc_accept: " << mc_obj->mc_get_error(result) << endl;
   }
   // make sure the connection is closed
   mc_obj->mc_close();
   return result;
}

//=====================================================================
int main(int, char *[]) {
   mc_obj = new mshn_com();
   msl_obj = new mshn_sl();
   core_cert_check = CERT_NONE;
```

```
    int result = msl_obj->mshn_sl_init("dummy file");
    if (result != MSHN_OK) {
        err_out = msl_obj->mshn_sl_show_error(result);
        cout << "mshn_sl_init " << err_out << endl;
        mshn_free(err_out, NULL);
    }else{
        // Get the shared symmetric key for encryption/decryption
        read_data(&sym_key, key_fname, BUFF_SIZE);

        do_register(msl_obj, core_id, core_cert, &passphrase,
                                    core_comm_sec, core_cert_check);

        // now receive and handle incoming scheduling requests
        while (result == MSHN_OK) {
                result = handle_request();
        }
    }
    return 0;
}
```

E. COMPUTE RESOURCE

```
//****************************************************
// File:  resource.cpp
// Name:  Roger Wright
//
// Project: MSHN
//
// Operating Environment: Windows 95/Windows NT
// Compiler: Borland C++ for Windows
// Date:  12 MAY 98
//
// Description: MSHN demonstration compute resource process
//****************************************************

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include "mshn_sl.h"
#include "mshn_mem.h"
#include "mshn_com.h"
#include "mshn_defs.h"
#include "mshn_err.h"
#include "mshn_types.h"
#include "commutil.h"
#include "mshn_demo.h"
#include "showutil.h"
```

```
char *err_out;
mshn_sl  *msl_obj;
mshn_com *mc_obj;
mshn_data sym_key;
mshn_data job_sess_key;
const int BUFF_SIZE = 1024;
cert_checking resource_cert_check;
comm_security resource_comm_sec;
char *passphrase;
mshn_data resource_cert, resource_id;
int request_id;

char dummy_in[80];

//======================================================================
// Encrypt the data to be transmitted to the client
int   do_encrypt_client(
                  mshn_data          *results,
                  mshn_data          *app_signature)
{
   int result = MSHN_OK;

   // Now prepare the inputs for encryption
   const int num_params = 2;
   mshn_data work_array[num_params];
   work_array[0].the_length = results->the_length;
   work_array[0].the_data   = results->the_data;
   work_array[1].the_length = app_signature->the_length;
   work_array[1].the_data   = app_signature->the_data;
   mshn_data *enc_array[num_params];
   for (int i = 0; i < num_params; i++) {
      enc_array[i] = (mshn_data *)
                     mshn_malloc(sizeof(mshn_data *), NULL);
   }

   int numEncBuff, bytesEnc;
   numEncBuff = num_params;

   result = msl_obj->mshn_sl_encrypt(&job_sess_key,
                                     MSHN_ALG_DES,
                                     work_array,
                                     num_params,
                                     enc_array,
                                     &numEncBuff,
                                     &bytesEnc);

   if (result != MSHN_OK) {
      err_out = msl_obj->mshn_sl_show_error(result);
      cout << "mshn_sl_encrypt " << err_out << endl;
      mshn_free(err_out, NULL);
   }else{
```

```
      // copy the decrypted buffers to the output buffers
      results->the_length            = enc_array[0]->the_length;
      results->the_data              = enc_array[0]->the_data;
      app_signature->the_length      = enc_array[1]->the_length;
      app_signature->the_data        = enc_array[1]->the_data;
   }
   return result;
}

//======================================================================
// Encrypt the data to be transmitted to the mshn core components
int   do_encrypt_core(
                  mshn_data          *resource_id,
                  mshn_data          *info,
                  mshn_data          *signature)

{
   int result = MSHN_OK;

   // Now prepare the inputs for encryption
   const int num_params = 3;
   mshn_data work_array[num_params];
   work_array[0].the_length = resource_id->the_length;
   work_array[0].the_data   = resource_id->the_data;
   work_array[1].the_length = info->the_length;
   work_array[1].the_data   = info->the_data;
   work_array[2].the_length = signature->the_length;
   work_array[2].the_data   = signature->the_data;
   mshn_data *enc_array[num_params];
   for (int i = 0; i < num_params; i++) {
      enc_array[i] = (mshn_data *)
                        mshn_malloc(sizeof(mshn_data *), NULL);
   }

   int numEncBuff, bytesEnc;
   numEncBuff = num_params;

   result = msl_obj->mshn_sl_encrypt(&job_sess_key,
                                     MSHN_ALG_DES,
                                     work_array,
                                     num_params,
                                     enc_array,
                                     &numEncBuff,
                                     &bytesEnc);

   if (result != MSHN_OK) {
      err_out = msl_obj->mshn_sl_show_error(result);
      cout << "mshn_sl_encrypt " << err_out << endl;
      mshn_free(err_out, NULL);
   }else{
```

```
   // copy the decrypted buffers to the output buffers
      resource_id->the_length = enc_array[0]->the_length;
      resource_id->the_data   = enc_array[0]->the_data;
      info->the_length        = enc_array[1]->the_length;
      info->the_data          = enc_array[1]->the_data;
      signature->the_length   = enc_array[2]->the_length;
      signature->the_data     = enc_array[2]->the_data;

   }

   return result;
}

//=================================================================
// Sign the data to be transmitted to the client
int   do_sign_client(
                 const mshn_data   *results,
                       mshn_data   *app_signature)
{
   int result = MSHN_OK;
   // Clear the output
   app_signature->the_length = 0;
   app_signature->the_data = NULL;

   // Now prepare the inputs for signing
   const int num_sign = 1;
   mshn_data work_array[num_sign];
   work_array[0].the_length = results->the_length;
   work_array[0].the_data   = results->the_data;
   result = msl_obj->mshn_sl_sign(MSHN_ALG_DES,
                              MSHN_ALG_MD5,
                              &job_sess_key,
                              NULL,
                              work_array,
                              num_sign,
                              app_signature);

   if (result != MSHN_OK) {
      err_out = msl_obj->mshn_sl_show_error(result);
      cout << "mshn_sl_sign " << err_out << endl;
      mshn_free(err_out, NULL);
   }

   return result;
}

//=================================================================
// Sign the data to be transmitted to the mshn core components
int   do_sign_core(
         const mshn_data   *resource_id,
```

```
        const mshn_data    *info,
        mshn_data          *signature)
{
   int result = MSHN_OK;
   // Clear the output
   signature->the_length = 0;
   signature->the_data = NULL;

   // Now prepare the inputs for signing
   const int num_sign = 3;
   mshn_data work_array[num_sign];
   work_array[0].the_length = sizeof(int);
   work_array[0].the_data   = (unsigned char *) &request_id;
   work_array[1].the_length = resource_id->the_length;
   work_array[1].the_data   = resource_id->the_data;
   work_array[2].the_length = info->the_length;
   work_array[2].the_data   = info->the_data;

   result = msl_obj->mshn_sl_sign(MSHN_ALG_DES,
                                  MSHN_ALG_MD5,
                                  &job_sess_key,
                                  NULL,
                                  work_array,
                                  num_sign,
                                  signature);

   if (result != MSHN_OK) {
      err_out = msl_obj->mshn_sl_show_error(result);
      cout << "mshn_sl_sign " << err_out << endl;
            mshn_free(err_out, NULL);
   }

   return result;
}

//=================================================================
// Verify/Decrypt the data received from the client
int   verify_request(
                  int            comm_sec,
                  mshn_data      &job_info,
                  mshn_data      &token_data,
                  mshn_data      &user_cert,
                  mshn_data      &signature)
{
   int result = MSHN_OK;
   const int num_params = 4;
   const int num_signed_params = 3;

   // Prepare the inputs for encryption/decryption/signature
   // verification
   mshn_data work_array[num_params];
```

```
work_array[0].the_length = job_info.the_length;
work_array[0].the_data   = job_info.the_data;
work_array[1].the_length = token_data.the_length;
work_array[1].the_data   = token_data.the_data;
work_array[2].the_length = user_cert.the_length;
work_array[2].the_data   = user_cert.the_data;
work_array[3].the_length = signature.the_length;
work_array[3].the_data   = signature.the_data;

mshn_data *dec_array[num_params];
for (int idx=0; idx<num_params; idx++) {
   dec_array[idx] = (mshn_data *)
                  mshn_malloc(sizeof(mshn_data *), NULL);
}

// Determine how to handle the inputs
// What decryption/signature verification is necessary?
comm_security c_sec = (comm_security) comm_sec;
if ((c_sec == COMSEC_CON) || (c_sec == COMSEC_BOTH)) {
   // We must do decryption
   int numdecBuff, bytesdec;
   numdecBuff = num_params;

   result = msl_obj->mshn_sl_decrypt(&sym_key, MSHN_ALG_DES,
                  work_array, num_params, dec_array,
                  &numdecBuff, &bytesdec);
   if (result != MSHN_OK) {
         err_out = msl_obj->mshn_sl_show_error(result);
         cout << "mshn_sl_decrypt " << err_out << endl;
         mshn_free(err_out, NULL);
   }else{
      // Prepare for the next steps, by putting the decrypted buffers
      // into work_array
      for (int idx=0; idx<num_params; idx++) {
         work_array[idx].the_length = dec_array[idx]->the_length;
         work_array[idx].the_data = dec_array[idx]->the_data;
      }
      // copy decrypted data back to original
      job_info.the_length   = work_array[0].the_length;
      job_info.the_data     = work_array[0].the_data;
      token_data.the_length = work_array[1].the_length;
      token_data.the_data   = work_array[1].the_data;
      user_cert.the_length  = work_array[2].the_length;
      user_cert.the_data    = work_array[2].the_data;
      signature.the_length  = work_array[3].the_length;
      signature.the_data    = work_array[3].the_data;
   }
}
if (result == MSHN_OK) {
   if ((c_sec == COMSEC_INT) || (c_sec == COMSEC_BOTH)) {
   // We must do signature verification
```

```
    // First verify the certificate
    mshn_data *certificate = &work_array[2];
    result = do_cert_check(msl_obj, resource_cert_check, certificate);

    // Now get the public key
          mshn_data public_key;
    if (result == MSHN_OK) {
      result = msl_obj->mshn_sl_get_public_key(
               certificate, &public_key);

      if (result != MSHN_OK) {
        err_out = msl_obj->mshn_sl_show_error(result);
        cout << "mshn_sl_get_public_key " << err_out << endl;
        mshn_free(err_out, NULL);
      }
    }
      // Now do signature verification
      if (result == MSHN_OK) {
        int sig_valid = 0;
        result = msl_obj->mshn_sl_sig_verify(MSHN_ALG_DSA,
                    MSHN_ALG_SHA,
                    &public_key,
                    work_array,
                    num_signed_params,
                    &work_array[3], // this one is the signature
                    &sig_valid);

      if (result != MSHN_OK) {
        err_out = msl_obj->mshn_sl_show_error(result);
        cout << "mshn_sl_sig_verify " << err_out << endl;
        mshn_free(err_out, NULL);
      }else{
        if (sig_valid == MSHN_FALSE) {
          result = MSHN_INVALID_SIG;
          err_out = msl_obj->mshn_sl_show_error(result);
          cout << "mshn_sl_sig_verify " << err_out << endl;
          mshn_free(err_out, NULL);
        }
      }
   }
  }
}

if (result == MSHN_OK) {
  // If we get here, the inputs have been decrypted and verified
  // so lets do something with them
  cout << "Job Request accepted." << endl;
}
  for (int idx=0; idx<num_params; idx++) {
  mshn_free(dec_array[idx], NULL);
}
```

```
   return result;
}

//==========================================================================
int handle_request()
{
   int result = MSHN_OK;

   // Now accept connections from the client/laptop
   mshn_com   *new_conn;
   result = mc_obj->mc_accept("Client",
                  PORT_CLIENT_RESOURCE, &new_conn);
   if (result == MSHN_COM_OK) {
      result = MSHN_OK;

#ifdef DEBUG_MSHN_COM
      cout << "Accepted connection" << endl;
      new_conn->mc_display(cout);
#endif

      int bytes_rec;
      mshn_data job_info, token_data, user_cert, signature;

      // get the input from the client/laptop
      result = recv_int_4_data(new_conn,
                        (int *) &resource_comm_sec,
                        "Communications Security Option",
                        &job_info,
                        "Job Info                     ",
                        &token_data,
                        "Security Token               ",
                        &user_cert,
                        "User Certificate             ",
                        &signature,
                        "Client Signature             ",
                         bytes_rec);

      if (result == MSHN_COM_OK) {
         result = MSHN_OK;
         cout << "Received (" << bytes_rec << ") bytes." << endl;

         // Now process the received data
         result = verify_request(resource_comm_sec,
                        job_info,
                        token_data,
                        user_cert,
                        signature);


         if (result == MSHN_OK) {
```

```
        // decrypt job session key

        // run the application

        // send output to client
        mshn_data results;
        mshn_data app_signature;
        mshn_token the_token;
        mshn_data_to_token(&token_data, &the_token);

        job_sess_key.the_length =
              the_token.job_session_key.the_length;
        job_sess_key.the_data    =
              the_token.job_session_key.the_data;

        request_id = the_token.request_id;

        // fill results with data just to test comms

        char *result_data = "This is sample job results";
        results.the_length = strlen(result_data);
        results.the_data = result_data;

        if ((resource_comm_sec == COMSEC_INT) ||
             (resource_comm_sec == COMSEC_BOTH)) {
           // We must do signature

           do_sign_client(&results, &app_signature);
        }

        if ((resource_comm_sec == COMSEC_CON) ||
             (resource_comm_sec == COMSEC_BOTH)) {
              // We must do encryption

           do_encrypt_client(&results, &app_signature);
        }

        int bytes_sent;
        send_2_data(new_conn,
                    &results,          "Results                ",
                    &app_signature, "Application Signature",
                     bytes_sent);

    }

}else{
        cout << "recv_int_4_data: "
             << new_conn->mc_get_error(result) << endl;
}
// make sure the connection is closed
new_conn->mc_close();
```

```
    }else{
        cout << "mc_accept: " << mc_obj->mc_get_error(result) << endl;
    }
    // make sure the connection is closed
    mc_obj->mc_close();
    return result;
}

//==================================================================
// send job status info to the resource status server
// if necessary, sign and encrypt data first
int update_rss()
{
    int result = MSHN_OK;
    mshn_data status_info;

    cout << "Sending status info to resource status server..." << endl;
    // Set up connection to the RSS

    result = mc_obj->mc_connect(IP_RSS, PORT_RESOURCE_RSS);
    if (result == MSHN_COM_OK) {
        int bytes_sent;
        cout << "Connection made" << endl;
        mc_obj->mc_display(cout);

        // create bogus status info
        char *stat_data = "This is sample job status data";
        status_info.the_length = strlen(stat_data);
        status_info.the_data = stat_data;

        // sign data before sending
        mshn_data signature;
        signature.the_data = NULL;
        signature.the_length = 0;

        mshn_data temp_resource_id;
        temp_resource_id.the_length = resource_id.the_length;
        temp_resource_id.the_data = (unsigned char *)
                mshn_malloc(resource_id.the_length, NULL);
        memcpy(temp_resource_id.the_data, resource_id.the_data,
                resource_id.the_length);

        if ((resource_comm_sec == COMSEC_INT) ||
            (resource_comm_sec == COMSEC_BOTH)) {
            // We must do signature

            do_sign_core(&temp_resource_id,
                         &status_info,
                         &signature);
        }
```

```
        if ((resource_comm_sec == COMSEC_CON) ||
             (resource_comm_sec == COMSEC_BOTH)) {
            // We must do encryption

            do_encrypt_core(&temp_resource_id,
                            &status_info,
                            &signature);
        }

        // send job status info to the resource status server
        result = send_int_3_data(mc_obj,
                                 request_id,
                                 "Request ID         ",
                                &temp_resource_id,
                                 "Resource ID        ",
                                &status_info,
                                 "Status Info        ",
                                &signature,
                                 "Resource Signature",
                                 bytes_sent);

        if (result == MSHN_COM_OK) {
            result = MSHN_OK;
            cout << "Sent (" << bytes_sent << ") bytes." << endl;

        }else{
            cout << "send_int_3_data: " << mc_obj->mc_get_error(result)
                 << endl;
        }

    }else{
        cout << "mc_connect: " << mc_obj->mc_get_error(result) << endl;
    }

    // make sure the connection is closed
    mc_obj->mc_close();

    return result;
}

//=====================================================================
// send job requirements info to the resource requirements database
// if necessary, sign and encrypt data first
int update_rrd()
{
    int result = MSHN_OK;
    mshn_data requirements_info;
    cout << "Sending requirements info to resource requirements
             database..." << endl;
    // Set up connection to the RRD
```

```
result = mc_obj->mc_connect(IP_RRD, PORT_RESOURCE_RRD);
if (result == MSHN_COM_OK) {
   int bytes_sent;
   cout << "Connection made" << endl;
   mc_obj->mc_display(cout);

   // create bogus requirements info
   char *req_data = "This is sample job requirements data";
   requirements_info.the_length = strlen(req_data);
   requirements_info.the_data = req_data;

   // sign data before sending
   mshn_data signature;
   signature.the_data = NULL;
   signature.the_length = 0;

   mshn_data temp_resource_id;
   temp_resource_id.the_length = resource_id.the_length;
   temp_resource_id.the_data = (unsigned char *)
         mshn_malloc(resource_id.the_length, NULL);
   memcpy(temp_resource_id.the_data, resource_id.the_data,
         resource_id.the_length);

   if ((resource_comm_sec == COMSEC_INT) ||
       (resource_comm_sec == COMSEC_BOTH)) {
      // We must do signature

       do_sign_core(&temp_resource_id,
                    &requirements_info,
                    &signature);
   }

   if ((resource_comm_sec == COMSEC_CON) ||
       (resource_comm_sec == COMSEC_BOTH)) {
      // We must do encryption
      do_encrypt_core(&temp_resource_id,
                      &requirements_info,
                      &signature);
   }

   // send job status info to the resource status server
   result = send_int_3_data(mc_obj,
                            request_id,
                           "Request ID          ",
                           &temp_resource_id,
                           "Resource ID         ",
                           &requirements_info,
                           "Requirements Info ",
                           &signature,
                           "Resource Signature",
                            bytes_sent);
```

```
        if (result == MSHN_COM_OK) {
            result = MSHN_OK;
            cout << "Sent (" << bytes_sent << ") bytes." << endl;

        }else{
            cout << "send_int_3_data: " << mc_obj->mc_get_error(result)
                 << endl;
        }

    }else{
        cout << "mc_connect: " << mc_obj->mc_get_error(result) << endl;
    }

    // make sure the connection is closed
    mc_obj->mc_close();

    return result;
}

//======================================================================
int main(int, char *[]) {

    mc_obj = new mshn_com();
    msl_obj = new mshn_sl();

    resource_cert_check = CERT_NONE;
    int result = msl_obj->mshn_sl_init("dummy file");
    if (result != MSHN_OK) {
        err_out = msl_obj->mshn_sl_show_error(result);
        cout << "mshn_sl_init " << err_out << endl;
        mshn_free(err_out, NULL);
    }else{
        // Get the shared symmetric key for encryption/decryption
        read_data(&sym_key, key_fname, BUFF_SIZE);

        do_register(msl_obj,
                    resource_id,
                    resource_cert,
                   &passphrase,
                    resource_comm_sec,
                    resource_cert_check);

        // now receive and handle incoming scheduling requests
        while (result == MSHN_OK) {
            result = handle_request();
            update_rss();
            update_rrd();
        }
    }
    return 0;
}
```

F. MSHN_COM.H

```
//****************************************************
//****************************************************
// File:  mshn_com.h
// Name:  David Shifflett
//
// Project: MSHN
//
// Operating Environment: Windows 95/Windows NT
// Compiler: Borland C++ for Windows
// Date: 18 MAY 98
//
// Description: TCP/IP socket communication services for MSHN components
//****************************************************

#ifndef _MSHN_COM_H
#define _MSHN_COM_H

#include <winsock.h>
#include <iostream.h>

// MSHN Communications layer class

// State definitions
#define MSHN_COM_OK                          -2001
#define MSHN_COM_CONNECTING                  -2002
#define MSHN_COM_CONNECTED                   -2003
#define MSHN_COM_BOUND                       -2004
#define MSHN_COM_ACCEPTING                   -2005
#define MSHN_COM_UNINITIALIZED               -2006

// State definitions
#define MSHN_UNINITIALIZED_STATE             20001
#define MSHN_INITIALIZED_STATE               20002
#define MSHN_BOUND_STATE                     20003
#define MSHN_ACCEPTING_STATE                 20004
#define MSHN_CONNECTING_STATE                20005
#define MSHN_CONNECTED_STATE                 20006

class mshn_com {

// Class data
protected:
   SOCKET         our_socket;
   int            our_port;
   char          *our_addr;
   int            our_state;
   int            socket_state;
```

```
public:
      mshn_com();

protected:
// This constructor is only to be used for 'accepted connections'
      mshn_com(
            SOCKET            the_socket,
            int               the_state);

public:
   virtual ~mshn_com();

// This function will create a socket and wait for an incoming
// connection on the specified port.
// The accepted connection will be available for sending/receiving
// on the returned 'mshn_com' object.
// The 'mshn_com' object that does the accept should not be used
// for sending/receiving, it should be used only for accepting
// more connections.
   int      mc_accept (
                  const char *label,
                  int         port,
                  mshn_com    **new_conn);

// This function will create a socket and wait until it can connect
// with a socket at the specified address, on the specified port.
// After successful connection the 'mshn_com' object can be used
// for sending/receiving.
   int      mc_connect (
                  const char *addr,
                  int         port);

// This function will send data via the connected socket.
// The number of bytes sent will be returned.
   int      mc_send (
                  const char *data,
                  int         len,
                  int        &bytes_sent);

// This function will receive data via the connected socket.
// The 'len' parameter should specify the size of the 'data' buffer.
// The number of bytes recevied will be returned.
   int      mc_recv (
                  char    *data,
                  int      len,
                  int     &bytes_received);

// This function will close the socket.
   int      mc_close ();

// This function will display the current object.
```

```
void mshn_com::mc_display (
                        ostream       &stream);

// This function will return a description of the error.
char *mshn_com::mc_get_error (
                        int         err_num);

protected:
// This function is used to set the class to a clean, uninitialized
state
void mc_cleanup();

};
#endif
```

G. MSHN_COM.CPP

```
//*******************************************************
//*******************************************************
// File:  mshn_com.cpp
// Name:  David Shifflett
//
// Project: MSHN
//
// Operating Environment: Windows 95/Windows NT
// Compiler: Borland C++ for Windows
// Date: 18 MAY 98
//
// Description: TCP/IP socket communication services for MSHN components
//*******************************************************

#include <stdio.h>
#include <dos.h>              // for sleep
#include "mshn_com.h"
#include "showutil.h"

// Default WinSock Version (1.1)
#define WS_VERSION_REQD 0x0101
WSADATA stWSAData;     // WinSock DLL Info

mshn_com::mshn_com()
{
   our_socket      = INVALID_SOCKET;
   our_port        = 0;
   our_addr        = NULL;
   our_state       = MSHN_UNINITIALIZED_STATE;
   socket_state    = MSHN_UNINITIALIZED_STATE;
}
```

```
mshn_com::mshn_com(
         SOCKET          the_socket,
         int             the_state)
{
   our_socket      = the_socket;
   our_port        = 0;
   our_addr        = NULL;
   our_state       = MSHN_INITIALIZED_STATE;
   socket_state    = the_state;
}

mshn_com::~mshn_com()
{
      mc_cleanup();
}

//======================================================================
// This function will create a socket and wait for an incoming
// connection on the specified port.
// The accepted connection will be available for sending/receiving
// on the returned 'mshn_com' object.
// The 'mshn_com' object that does the accept should not be used
// for sending/receiving, it should be used only for accepting
// more connections.
int mshn_com::mc_accept (
                        const char *label,
                        int         port,
                        mshn_com    **new_conn)
{
   int result = MSHN_COM_OK;
   *new_conn = NULL;
#if defined (DEBUG_MSHN_COM1)
      cout << "Accepting connections on port: " << port << endl;
#endif

   // Make sure we are in the right state
   if (our_socket != INVALID_SOCKET) {
      switch (socket_state) {
            case MSHN_BOUND_STATE: {
                  result = MSHN_COM_BOUND;
            break;
            }
            case MSHN_ACCEPTING_STATE: {
                  result = MSHN_COM_ACCEPTING;
            break;
            }
            case MSHN_CONNECTING_STATE: {
                  result = MSHN_COM_CONNECTING;
            break;
            }
            case MSHN_CONNECTED_STATE: {
```

```
                    result = MSHN_COM_CONNECTED;
                break;
                }
        }
        mc_cleanup();
    }

// Make sure it is ok to proceed, and that we aren't already accepting
    if (result == MSHN_COM_OK) {
        if (our_state == MSHN_UNINITIALIZED_STATE) {
                    // Initialize Windows Sockets DLL
                    result = WSAStartup(WS_VERSION_REQD, &stWSAData);
                    // WSAStartup() returns error value if failed
                    // (0 on success)
                    if (result == 0) result = MSHN_COM_OK;
                if (result == MSHN_COM_OK) {
#if defined (DEBUG_MSHN_COM)
        cout << "wVersion " << stWSAData.wVersion << endl;
        cout << "wHighVersion " << stWSAData.wHighVersion << endl;
        cout << "szDescription " << stWSAData.szDescription << endl;
        cout << "szSystemStatus " << stWSAData.szSystemStatus << endl;
        cout << "iMaxSockets " << stWSAData.iMaxSockets << endl;
        cout << "iMaxUdpDg " << stWSAData.iMaxUdpDg << endl;
        cout << "lpVendorInfo " << stWSAData.lpVendorInfo << endl;
#endif
                    our_state = MSHN_INITIALIZED_STATE;
            }
        }
    }

    if (result == MSHN_COM_OK) {
        // Get a TCP socket
#if defined (DEBUG_MSHN_COM)
        cout << "About to create a socket" << endl;
#endif
        our_socket = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (our_socket == INVALID_SOCKET) {
                result = WSAGetLastError();
            mc_cleanup();
#if defined (DEBUG_MSHN_COM)
        cout << "Invalid socket: " << result << endl;
#endif
        }else{
#if defined (DEBUG_MSHN_COM)
        cout << "socket (" << our_socket << ")" << endl;
#endif
        // save the inputs in our class
        our_port = port;
        socket_state = MSHN_ACCEPTING_STATE;

        // Now bind the socket
```

```
      struct sockaddr the_addr;
      int namelen = sizeof(the_addr);
      u_short temp_port = htons(our_port);
      memset(the_addr.sa_data, 0, sizeof(the_addr.sa_data));
      memcpy(the_addr.sa_data, &temp_port, sizeof(temp_port));
      the_addr.sa_family = AF_INET;
      result = bind(our_socket, &the_addr, namelen);
      if (result == 0) result = MSHN_COM_OK;
         if (result != MSHN_COM_OK) {
            result = WSAGetLastError();
            mc_cleanup();
#if defined (DEBUG_MSHN_COM)
      cout << "bind returns: " << result << endl;
#endif
      }else{
            socket_state = MSHN_BOUND_STATE;
           }
      }
   }

   // Make sure it is ok to proceed
   if (result == MSHN_COM_OK) {
      // make our socket non-blocking
      u_long lOnOff = TRUE;
      result = ioctlsocket(our_socket, FIONBIO, &lOnOff);
      if (result == 0) result = MSHN_COM_OK;
      if (result != MSHN_COM_OK) {
         result = WSAGetLastError();
         mc_cleanup();
#if defined (DEBUG_MSHN_COM)
      cout << "ioctlsocket returns: " << result << endl;
#endif
      }
   }

   // Make sure it is ok to proceed with the accepting
   if (result == MSHN_COM_OK) {
      // Now its time to listen for connections
      result = listen(our_socket, 0);
      if (result == 0) result = MSHN_COM_OK;
      if (result != MSHN_COM_OK) {
         result = WSAGetLastError();
         mc_cleanup();
#if defined (DEBUG_MSHN_COM)
      cout << "listen returns: " << result << endl;
#endif
      }else{
         SOCKET child_socket;
         struct sockaddr child_addr;
         int child_len = sizeof(child_addr);
         socket_state = MSHN_ACCEPTING_STATE;
```

```
        // Display the label to tell the user what is happening
        cout << endl << "Accepting connections from " << label
             << " on port " << port << endl;

#if defined (DEBUG_MSHN_COM)
      cout << "About to accept" << endl;
#endif
        do {
           child_socket = accept(our_socket, &child_addr, &child_len);
           if (child_socket == INVALID_SOCKET) {
                 result = WSAGetLastError();
                             if (result == WSAEWOULDBLOCK) {
#if defined (DEBUG_MSHN_COM)
      cout << "sleeping" << endl;
#endif
                 sleep(2);
           }else{
#if defined (DEBUG_MSHN_COM)
      cout << result << endl;
#endif
                       //
                       // Need to do something about other errors here
                       //
                 mc_cleanup();
              }
           }else{
#if defined (DEBUG_MSHN_COM)
      cout << "Accept returns (" << child_socket << ")" << endl;
#endif
              result = MSHN_COM_OK;
              *new_conn = new mshn_com(child_socket,
                          MSHN_CONNECTED_STATE);
           }
        } while (result == WSAEWOULDBLOCK);
     }
  }

  return result;
}

//======================================================================
// This function will create a socket and wait until it can connect
// with a socket at the specified address, on the specified port.
// After successful connection the 'mshn_com' object can be used
// for sending/receiving.
int mshn_com::mc_connect (
                 const char *addr,
                 int         port)
{
     int result = MSHN_COM_OK;
```

```
#if defined (DEBUG_MSHN_COM)
      cout << "Attempting a connection to (" << addr << ") on port ("
           << port << ")." << endl;
#endif

   // Make sure we are in the right state
   if ((our_socket != INVALID_SOCKET) ||
       (socket_state != MSHN_UNINITIALIZED_STATE)) {
      switch (socket_state) {
            case MSHN_BOUND_STATE: {
                  result = MSHN_COM_BOUND;
            break;
            }
            case MSHN_ACCEPTING_STATE: {
                  result = MSHN_COM_ACCEPTING;
            break;
            }
            case MSHN_CONNECTING_STATE: {
                  result = MSHN_COM_CONNECTING;
            break;
            }
            case MSHN_CONNECTED_STATE: {
                  result = MSHN_COM_CONNECTED;
            break;
            }
         }
            mc_cleanup();
      }

   // Make sure it is ok to proceed
   if (result == MSHN_COM_OK) {
      if (our_state == MSHN_UNINITIALIZED_STATE) {
            // Initialize Windows Sockets DLL
            result = WSAStartup(WS_VERSION_REQD, &stWSAData);
            // WSAStartup() returns error value if failed (0 on success)
            if (result == 0) result = MSHN_COM_OK;
            if (result == MSHN_COM_OK) {
#if defined (DEBUG_MSHN_COM)
      cout << "wVersion " << stWSAData.wVersion << endl;
      cout << "wHighVersion " << stWSAData.wHighVersion << endl;
      cout << "szDescription " << stWSAData.szDescription << endl;
      cout << "szSystemStatus " << stWSAData.szSystemStatus << endl;
      cout << "iMaxSockets " << stWSAData.iMaxSockets << endl;
      cout << "iMaxUdpDg " << stWSAData.iMaxUdpDg << endl;
      cout << "lpVendorInfo " << stWSAData.lpVendorInfo << endl;
#endif
                  our_state = MSHN_INITIALIZED_STATE;
         }
      }
   }
```

```
   // Make sure it is ok to proceed
   if (result == MSHN_COM_OK) {
            // Get a TCP socket
            our_socket = socket (AF_INET, SOCK_STREAM, 0);
      if (our_socket == INVALID_SOCKET) {
            result = WSAGetLastError();
                  mc_cleanup();
#if defined (DEBUG_MSHN_COM)
      cout << "Invalid socket: " << result << endl;
#endif
      }else{
            socket_state = MSHN_CONNECTING_STATE;
#if defined (DEBUG_MSHN_COM)
      cout << "socket (" << our_socket << ")" << endl;
#endif
      }
   }

   // Make sure it is ok to proceed
   if (result == MSHN_COM_OK) {
      // save the inputs in our class
      our_port = port;
      our_addr = strdup(addr);

      struct sockaddr_in the_addr_in;
      memset(&the_addr_in, 0, sizeof(the_addr_in));
      the_addr_in.sin_port = htons(our_port);
      the_addr_in.sin_family = AF_INET;
      the_addr_in.sin_addr.s_addr = inet_addr(our_addr);

      int the_len = sizeof(the_addr_in);

      // Now its time to make a connection
#if defined (DEBUG_MSHN_COM)
      cout << "About to connect" << endl;
#endif
      result = connect(our_socket, (struct sockaddr *)
                        &the_addr_in, the_len);
      if (result == 0) result = MSHN_COM_OK;
      if (result != MSHN_COM_OK) {
         result = WSAGetLastError();
         mc_cleanup();
#if defined (DEBUG_MSHN_COM)
      cout << "connect returns: " << result << endl;
#endif
      }else{
         socket_state = MSHN_CONNECTED_STATE;
      }
   }
   return result;
}
```

```
//======================================================================
// This function will send data via the connected socket.
// The number of bytes sent will be returned.
int mshn_com::mc_send (
            const char *data,
            int         len,
            int        &bytes_sent)
{
   int result = MSHN_COM_OK;
   bytes_sent = 0;

   // Make sure we are in the right state
   if (socket_state != MSHN_CONNECTED_STATE) {
      switch (socket_state) {
            case MSHN_BOUND_STATE: {
                  result = MSHN_COM_BOUND;
            break;
            }
            case MSHN_ACCEPTING_STATE: {
                  result = MSHN_COM_ACCEPTING;
            break;
            }
            case MSHN_CONNECTING_STATE: {
                  result = MSHN_COM_CONNECTING;
            break;
            }
            case MSHN_UNINITIALIZED_STATE: {
                  result = MSHN_COM_UNINITIALIZED;
            break;
            }
      }
      mc_cleanup();
   }

   // Make sure it is ok to proceed
   if (result == MSHN_COM_OK) {
#if defined (DEBUG_MSHN_COM)
      cout << "Sending (" << len << ") bytes." << endl;
#endif
      result = send(our_socket, data, len, 0);
      if (result < 0) {
            result = WSAGetLastError();
            mc_cleanup();
#if defined (DEBUG_MSHN_COM)
      cout << "send returns: " << result << endl;
#endif
      }else{
#if defined (DEBUG_MSHN_COM)
      cout << "Sent (" << result << ") bytes." << endl;
#endif
      bytes_sent = result;
```

```
      result = MSHN_COM_OK;
      }
   }
   return result;
}

//==================================================================
// This function will receive data via the connected socket.
// The 'len' parameter should specify the size of the 'data' buffer.
// The number of bytes recevied will be returned.
int mshn_com::mc_recv (
                  char          *data,
                  int            len,
                  int           &bytes_received)
{
   int result = MSHN_COM_OK;
   bytes_received = 0;

   // Make sure we are in the right state
   if (socket_state != MSHN_CONNECTED_STATE) {
      switch (socket_state) {
            case MSHN_BOUND_STATE: {
                  result = MSHN_COM_BOUND;
            break;
            }
            case MSHN_ACCEPTING_STATE: {
                  result = MSHN_COM_ACCEPTING;
            break;
            }
            case MSHN_CONNECTING_STATE: {
                  result = MSHN_COM_CONNECTING;
            break;
            }
            case MSHN_UNINITIALIZED_STATE: {
                  result = MSHN_COM_UNINITIALIZED;
            break;
            }
      }
      mc_cleanup();
   }

   // Make sure it is ok to proceed
   if (result == MSHN_COM_OK) {
#if defined (DEBUG_MSHN_COM)
      cout << "Waiting to receive (" << len << ") bytes." << endl;
#endif
      // Now read some data from the other side
         do {
            result = recv(our_socket, data, len, 0);
               if (result < 0) {
                  result = WSAGetLastError();
```

```
#if defined (DEBUG_MSHN_COM)
      cout << "recv error: " << result;
      if (result == WSAEWOULDBLOCK) cout << ", WSAEWOULDBLOCK";
      cout << ", sleeping" << endl;
#endif
                  sleep(1);
               }else{
#if defined (DEBUG_MSHN_COM)
   show_pointer((uint8 *)data, result, "Received bytes");
#endif
                        bytes_received = result;
                        result = MSHN_COM_OK;
                  }
         } while (result == WSAEWOULDBLOCK);
      }
   return result;
}

//===================================================================
// This function will close the socket.
int mshn_com::mc_close ()
{
   int result = MSHN_COM_OK;
   if (our_socket != INVALID_SOCKET) {
      closesocket(our_socket);
#if defined (DEBUG_MSHN_COM)
   cout << "Closed socket (" << our_socket << ")." << endl;
#endif
   }
   our_socket = INVALID_SOCKET;
   socket_state = MSHN_UNINITIALIZED_STATE;
   return result;
}

//===================================================================
// This function will display the current object.
void mshn_com::mc_display (
                        ostream       &stream)
{
   char *state_char;

   if (our_socket == INVALID_SOCKET) {
      stream << "We don't have a socket." << endl;
   }else{
      stream << "Our socket is (" << our_socket << ")." << endl;
   }
   stream << "Our port is (" << our_port << ")." << endl;
      if (our_addr == NULL) {
            stream << "We don't have an address." << endl;
   }else{
      stream << "Our address is (" << our_addr << ")." << endl;
```

```
    }
    switch (socket_state) {
          case MSHN_UNINITIALIZED_STATE: {
             state_char = "Uninitialized";
          break;
          }
          case MSHN_BOUND_STATE: {
             state_char = "Bound";
          break;
          }
          case MSHN_ACCEPTING_STATE: {
             state_char = "Accepting";
          break;
          }
          case MSHN_CONNECTING_STATE: {
             state_char = "Connecting";
          break;
          }
          case MSHN_CONNECTED_STATE: {
             state_char = "Connected";
          break;
          }
       }
    stream << "Our current state is (" << state_char << ")."
           << endl << endl;
}

//======================================================================
// This function will return the MSHN_COM error condition.
char *mshn_com::mc_get_error (
                        int                 err_num)
{
    char *result = NULL;
    switch (err_num) {
       case MSHN_COM_OK: {
             result = strdup("No error");
             break;
       }
       case MSHN_COM_CONNECTING: {
             result = strdup("MSHN COM is connecting,
             this is an invalid state for the desired operation.");
             break;
       }
       case MSHN_COM_CONNECTED: {
             result = strdup("MSHN COM is connected, this is an
             invalid state for the desired operation.");
             break;
       }
       case MSHN_COM_BOUND: {
             result = strdup("MSHN COM is bound, this is an
             invalid state for the desired operation.");
```

```
            break;
        }
        case MSHN_COM_ACCEPTING: {
            result = strdup("MSHN COM is accepting, this is an
            invalid state for the desired operation.");
            break;
        }
        case MSHN_COM_UNINITIALIZED: {
            result = strdup("MSHN COM is uninitialized, this is an
            invalid state for the desired operation.");
            break;
        }
        default: {
            result = (char *)malloc(80);
            sprintf(result, "Unknown error (%d)", err_num);
            break;
        }
    }
    return result;
}

//=====================================================================
// This function is used to set the class to a clean,
// uninitialized state
void mshn_com::mc_cleanup()
{
   // close the socket
   mc_close();
   if (our_addr != NULL) {
      delete(our_addr);
   }
   if (our_state == MSHN_INITIALIZED_STATE) {
         // say good-by to WinSock DLL
      WSACleanup();
         our_state = MSHN_UNINITIALIZED_STATE;
   }
}
```

H. COMMUTIL.H

```
//*****************************************************
// File:  commutil.h
// Name:  David Shifflett & Roger Wright
//
// Project: MSHN
//
// Operating Environment: Windows 95/Windows NT
// Compiler: Borland C++ for Windows
// Date: 18 MAY 98
```

```
//
// Description: MSHN communication utility functions
//*****************************************************

#ifndef _COMMUTIL_H
#define _COMMUTIL_H

#include "mshn_types.h"
#include "mshn_com.h"
#include "mshn_sl.h"
#include "mshn_defs.h"


// This function will read from the specified file into the
// specified mshn_data type
int    read_data(
            mshn_data               *new_data,
            const char              *filename,
            int                      buff_size);

// This function will write the specified data to the specified file
int    write_data(
                  const mshn_data   *new_data,
                  const char        *filename);

// This function will write the specified data to the REQUEST DB
int    add_to_req_db(
                  const char *the_fname,
                  const mshn_data *user_id,
                  int comm_sec,
                  int request_id,
                  const mshn_data *sess_key);

// This function will find and return the REQUEST DB info
// for the specified request_id
int    get_from_req_db(
                  const char   *the_fname,
                  int          desired_req_id,
                  mshn_data    *user_id,
                  int          &comm_sec,
                  mshn_data    *sess_key);

// This function will bundle up two mshn_data types and send
// them out the supplied connection.
int    send_2_data(
                  mshn_com          *mc_obj,
                  const mshn_data   *dt1,
                  const char        *dt1_label,
                  const mshn_data   *dt2,
                  const char        *dt2_label,
                  int               &bytes_sent);
```

```
// This function will bundle up three mshn_data types and send
// them out the supplied connection.
int    send_3_data(
                    mshn_com           *mc_obj,
                    const mshn_data    *dt1,
                    const char         *dt1_label,
                    const mshn_data    *dt2,
                    const char         *dt2_label,
                    const mshn_data    *dt3,
                    const char         *dt3_label,
                    int                &bytes_sent);

// This function will receive data from the supplied connection and
// break it into two mshn_data types.
int    recv_2_data(
                    mshn_com     *mc_obj,
                    mshn_data    *dt1,
                    char         *dt1_label,
                    mshn_data    *dt2,
                    char         *dt2_label,
                    int          &bytes_rec);

// This function will receive data from the supplied connection and
// break it into three mshn_data types.
int    recv_3_data(
                    mshn_com     *mc_obj,
                    mshn_data    *dt1,
                    char         *dt1_label,
                    mshn_data    *dt2,
                    char         *dt2_label,
                    mshn_data    *dt3,
                    char         *dt3_label,
                    int          &bytes_rec);

// This function will bundle up an integer and 3 mshn_data types and
// send them out the supplied connection.
int    send_int_3_data(
                    mshn_com           *mc_obj,
                    int                it1,
                    const char         *it1_label,
                    const mshn_data    *dt1,
                    const char         *dt1_label,
                    const mshn_data    *dt2,
                    const char         *dt2_label,
                    const mshn_data    *dt3,
                    const char         *dt3_label,
                    int                &bytes_sent);

// This function will receive data from the supplied connection and
// break it into an integer and 3 mshn_data types.
```

```
int    recv_int_3_data(
                        mshn_com      *mc_obj,
                        int           *it1,
                        char          *it1_label,
                        mshn_data     *dt1,
                        char          *dt1_label,
                        mshn_data     *dt2,
                        char          *dt2_label,
                        mshn_data     *dt3,
                        char          *dt3_label,
                        int           &bytes_rec);

// This function will bundle up an integer and 4 mshn_data types and
// send them out the supplied connection.
int    send_int_4_data(
                        mshn_com            *mc_obj,
                        int                 it1,
                        const char          *it1_label,
                        const mshn_data     *dt1,
                        const char          *dt1_label,
                        const mshn_data     *dt2,
                        const char          *dt2_label,
                        const mshn_data     *dt3,
                        const char          *dt3_label,
                        const mshn_data     *dt4,
                        const char          *dt4_label,
                        int                 &bytes_sent);

// This function will receive data from the supplied connection and
// break it into an integer and 4 mshn_data types.
int    recv_int_4_data(
                        mshn_com            *mc_obj,
                        int                 *it1,
                        char                *it1_label,
                        mshn_data           *dt1,
                        char                *dt1_label,
                        mshn_data           *dt2,
                        char                *dt2_label,
                        mshn_data           *dt3,
                        char                *dt3_label,
                        mshn_data           *dt4,
                        char                *dt4_label,
                        int                 &bytes_rec);

// This function will bundle up 6 mshn_data types and send
// them out the supplied connection.
int    send_6_data(
                        mshn_com            *mc_obj,
                        const mshn_data     *dt1,
                        const char          *dt1_label,
                        const mshn_data     *dt2,
```

```
                        const char          *dt2_label,
                        const mshn_data     *dt3,
                        const char          *dt3_label,
                        const mshn_data     *dt4,
                        const char          *dt4_label,
                        const mshn_data     *dt5,
                        const char          *dt5_label,
                        const mshn_data     *dt6,
                        const char          *dt6_label,
                        int                 &bytes_sent);

// This function will receive data from the supplied connection and
// break it into 6 mshn_data types.
int   recv_6_data(
                        mshn_com     *mc_obj,
                        mshn_data    *dt1,
                        char         *dt1_label,
                        mshn_data    *dt2,
                        char         *dt2_label,
                        mshn_data    *dt3,
                        char         *dt3_label,
                        mshn_data    *dt4,
                        char         *dt4_label,
                        mshn_data    *dt5,
                        char         *dt5_label,
                        mshn_data    *dt6,
                        char         *dt6_label,
                        int          &bytes_rec);

// Convert from a mshn_token structure to a mshn_data structure
void  token_to_mshn_data (
                        const mshn_token   the_token,
                        mshn_data          *the_data);

// Convert from a mshn_data structure to a mshn_token structure
void  mshn_data_to_token (
                  const mshn_data  *the_data,
                        mshn_token *the_token);

//  obtains the user ID, certificate, passphrase, communications
//  security option and certificate validation level from the user
//
int do_register(mshn_sl *msl_obj,
                mshn_data &user_id,
                mshn_data &user_cert,
                char **passphrase,
                comm_security &com_sec_option,
                cert_checking &cert_valid_level);

// checks the given certificate for authenticity
int do_cert_check(mshn_sl *the_sl,
```

```
                  cert_checking c_check,
                  mshn_data *the_cert);

#endif
```

I. COMMUTIL.CPP

```
//  Written by David Shifflett and Roger Wright
//
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include <conio.h>
#include "mshn_err.h"
#include "mshn_mem.h"
#include "commutil.h"

const int BUFF_SIZE = 1024;

//======================================================================
// Internal function for copying integers to a buffer
void add_int(unsigned char **new_buff_ptr, int the_int)
{
      memcpy(*new_buff_ptr, &the_int, sizeof(the_int));
      *new_buff_ptr += sizeof(the_int);
}

//======================================================================
// Internal function for copying mshn_data structures to a buffer
void add_mshn_data(unsigned char **new_buff_ptr,
                   const mshn_data *the_data)
{
   memcpy(*new_buff_ptr, &the_data->the_length,
      sizeof(the_data->the_length));
   *new_buff_ptr += sizeof(the_data->the_length);
   if (the_data->the_length > 0) {
      memcpy(*new_buff_ptr, the_data->the_data, the_data->the_length);
      *new_buff_ptr += the_data->the_length;
   }
}

//======================================================================
// Internal function for copying integers from a buffer
void get_int(unsigned char **new_buff_ptr, int *the_int)
{
      memcpy(the_int, *new_buff_ptr, sizeof(the_int));
      *new_buff_ptr += sizeof(the_int);
}
```

```
//============================================================================
// Internal function for copying mshn_data structures from a buffer
void get_mshn_data(unsigned char **new_buff_ptr, mshn_data *the_data)
{
   memcpy(&the_data->the_length, *new_buff_ptr,
      sizeof(the_data->the_length));
   *new_buff_ptr += sizeof(the_data->the_length);
   if (the_data->the_length > 0) {
      the_data->the_data = (unsigned char *)
                           mshn_malloc(the_data->the_length, NULL);
      memcpy(the_data->the_data, *new_buff_ptr, the_data->the_length);
      *new_buff_ptr += the_data->the_length;
   }else{
      the_data->the_data = NULL;
   }
}

//============================================================================
// This function will read from the specified file into the specified
// mshn_data type
int read_data(mshn_data *new_data, const char *filename, int buff_size)
{
   int result = MSHN_OK;
   // Clean out the outputs
   new_data->the_data = NULL;
   new_data->the_length = 0;
   new_data->the_data = (unsigned char *)mshn_malloc(buff_size, NULL);
   // Now read the key from a file
   FILE *inputFile;
      int idx=0;
   if ((inputFile = fopen(filename, "rb")) != NULL)
   {
// cout << "Read from file" << endl;
      while (!feof(inputFile)) {
            new_data->the_data[idx] = fgetc(inputFile);
// cout << " " << (int) new_data->the_data[idx];
// if ((idx>0) && ((idx % 19) == 0)) cout << endl;
            idx++;
            if (idx == buff_size) {
                  cout << "buffer overflow on read, truncating at "
                        << idx << " bytes" << endl;
            break;
         }
      }
   }
   fclose(inputFile);
   new_data->the_length = idx - 1;  // ignore eof byte
// cout << endl << "read length is " << new_data->the_length << endl;
   return result;
}
```

```
//==================================================================
// This function will write the specified data to the specified file
int write_data(const mshn_data *new_data, const char *filename)
{
   int result = MSHN_OK;
   // Now write the data to a file
   FILE *outputFile;
   if ((outputFile = fopen(filename, "wb")) != NULL)
   {
// cout << "Write to file" << endl;
      for (int idx = 0; idx< new_data->the_length; idx++) {
            fputc(new_data->the_data[idx], outputFile);
// cout << " " << (int) new_data->the_data[idx];
// if ((idx>0) && ((idx % 19) == 0)) cout << endl;
      }
   }
   fclose(outputFile);
// cout << endl << "write length is " << new_data->the_length << endl;
   return result;
}

//==================================================================
// This function will write the specified data to the REQUEST DB
// The format of the REQUEST DB will be:
//   comm_sec,request_id,
//   'length of user id'user_id
//   'length of sess_key'sess_key
//
int add_to_req_db(const char        *the_fname,
                  const mshn_data   *user_id,
                  int               comm_sec,
                  int               request_id,
                  const mshn_data   *sess_key)
{
   int result = MSHN_OK;
   // Now write the data to a file
   FILE *outputFile;
   if ((outputFile = fopen(the_fname, "ab")) != NULL)
   {
      fprintf(outputFile, "%d,%d,%d", comm_sec, request_id,
         user_id->the_length);
// cout << "Wrote " << comm_sec << " " << request_id << " "
        << user_id->the_length << endl;
      for (int idx = 0; idx< user_id->the_length; idx++) {
            fputc(user_id->the_data[idx], outputFile);
            }
            fprintf(outputFile, "%d", sess_key->the_length);
// cout << "Wrote " << sess_key->the_length << endl;
      for (int idx = 0; idx< sess_key->the_length; idx++) {
            fputc(sess_key->the_data[idx], outputFile);
            }
```

```
   }
   fclose(outputFile);
   return result;
}

//=====================================================================
// This function will find and return the REQUEST DB info
// for the specified request_id
int get_from_req_db(const char *the_fname, int desired_req_id,
                                mshn_data *user_id, int &comm_sec,
                                mshn_data *sess_key)
{
   int result = MSHN_OK;
   int request_id, c_sec;
   unsigned int temp_len;

   // Clean out the outputs
   user_id->the_data = NULL;
   user_id->the_length = 0;
   sess_key->the_data = NULL;
   sess_key->the_length = 0;
   comm_sec = -1;

   FILE *inputFile;
      int found=0;
   if ((inputFile = fopen(the_fname, "rb")) != NULL)
   {
      while ((!found) && (!feof(inputFile))) {
            fscanf(inputFile, "%d,%d,%d", &c_sec, &request_id,
                   &temp_len);
// cout << "Read " << c_sec << " " << request_id << " " << temp_len
        << endl;
         if (request_id == desired_req_id) {
            // we found it, read the data, and output it
            found = 1;
            comm_sec = c_sec;
            user_id->the_length = temp_len;
            user_id->the_data = (unsigned char *)
                                  mshn_malloc(temp_len, NULL);
            for (int idx = 0; idx< temp_len; idx++) {
                  user_id->the_data[idx] = fgetc(inputFile);
            }
            fscanf(inputFile, "%d", &temp_len);
// cout << "Read " << temp_len << endl;
            sess_key->the_length = temp_len;
            sess_key->the_data = (unsigned char *)
                                   mshn_malloc(temp_len, NULL);
                  for (int idx = 0; idx< temp_len; idx++) {
                        sess_key->the_data[idx] = fgetc(inputFile);
                  }
         }else{
```

```
            // simply skip to the next record
                for (int idx = 0; idx< temp_len; idx++) {
                    fgetc(inputFile);
                }
                fscanf(inputFile, "%d", &temp_len);
                for (int idx = 0; idx< temp_len; idx++) {
                    fgetc(inputFile);
                }
        }
      }
   }
   fclose(inputFile);
   if (!found) result = -1;
   return result;
}

//=====================================================================
// This function will bundle up two mshn_data types and send
// them out the supplied connection.
int   send_2_data(
                mshn_com            *mc_obj,
                const mshn_data     *dt1,
                const char          *dt1_label,
                const mshn_data     *dt2,
                const char          *dt2_label,
                int                 &bytes_sent)
{
   int result = MSHN_COM_OK;
   int total_length = dt1->the_length + dt2->the_length +
                                       sizeof(unsigned int) * 2;
   cout << "total length " << total_length << endl;
   mshn_data new_buff;
   new_buff.the_length = total_length;
   new_buff.the_data = (unsigned char *)
                    mshn_malloc(total_length, NULL);
   unsigned char *new_buff_ptr = new_buff.the_data;
   add_mshn_data(&new_buff_ptr, dt1);
   add_mshn_data(&new_buff_ptr, dt2);

   cout << "Sending everything: " << new_buff.the_length
        << " bytes." << endl;
   result = mc_obj->mc_send(new_buff.the_data, new_buff.the_length,
            bytes_sent);

   if (result == MSHN_COM_OK) {
      cout << "Sent (" << bytes_sent << ") bytes." << endl;
      cout << dt1_label <<  " length (" << dt1->the_length << ") bytes."
           << endl;
      cout << dt2_label <<  " length (" << dt2->the_length << ") bytes."
           << endl;
   }else{
```

```
      cout << "mc_send: " << mc_obj->mc_get_error(result) << endl;
   }
   mshn_free(new_buff.the_data, NULL);
   return result;
}

//======================================================================
// This function will bundle up three mshn_data types and send
// them out the supplied connection.
int   send_3_data(
                  mshn_com          *mc_obj,
                  const mshn_data   *dt1,
                  const char        *dt1_label,
                  const mshn_data   *dt2,
                  const char        *dt2_label,
                  const mshn_data   *dt3,
                  const char        *dt3_label,
                  int               &bytes_sent)
{
   int result = MSHN_COM_OK;
   int total_length = dt1->the_length + dt2->the_length
                    + dt3->the_length + sizeof(unsigned int) * 3;
   cout << "total length " << total_length << endl;
   mshn_data new_buff;
   new_buff.the_length = total_length;
   new_buff.the_data = (unsigned char *)
                        mshn_malloc(total_length, NULL);
   unsigned char *new_buff_ptr = new_buff.the_data;
   add_mshn_data(&new_buff_ptr, dt1);
   add_mshn_data(&new_buff_ptr, dt2);
   add_mshn_data(&new_buff_ptr, dt3);

   cout << "Sending everything: " << new_buff.the_length << " bytes."
        << endl;
   result = mc_obj->mc_send(new_buff.the_data, new_buff.the_length,
            bytes_sent);

   if (result == MSHN_COM_OK) {
      cout << "Sent (" << bytes_sent << ") bytes." << endl;
      cout << dt1_label << " length (" << dt1->the_length << ") bytes."
           << endl;
      cout << dt2_label << " length (" << dt2->the_length << ") bytes."
           << endl;
      cout << dt3_label << " length (" << dt3->the_length << ") bytes."
           << endl;
   }else{
      cout << "mc_send: " << mc_obj->mc_get_error(result) << endl;
   }
   mshn_free(new_buff.the_data, NULL);
   return result;
}
```

```
//===================================================================
// This function will receive data from the supplied connection and
// break it into two mshn_data types.
int   recv_2_data(
                  mshn_com     *mc_obj,
                  mshn_data    *dt1,
                  char         *dt1_label,
                  mshn_data    *dt2,
                  char         *dt2_label,
                  int          &bytes_rec)
{
   int result = MSHN_COM_OK;
   // Clean out the outputs
   dt1->the_data = NULL;
   dt2->the_data = NULL;
   dt1->the_length = 0;
   dt2->the_length = 0;
   mshn_data new_buff;

   new_buff.the_data = (unsigned char *)
                        mshn_malloc(BUFF_SIZE * 16, NULL);
   new_buff.the_length = BUFF_SIZE * 16;
   result = mc_obj->mc_recv(new_buff.the_data, new_buff.the_length,
            bytes_rec);

   unsigned char *new_buff_ptr = new_buff.the_data;
   get_mshn_data(&new_buff_ptr, dt1);
   get_mshn_data(&new_buff_ptr, dt2);

   if (result == MSHN_COM_OK) {
      cout << "Received (" << bytes_rec << ") bytes." << endl;
      cout << dt1_label << " length (" << dt1->the_length << ") bytes."
           << endl;
      cout << dt2_label << " length (" << dt2->the_length << ") bytes."
           << endl;
   }else{
      cout << "mc_recv: " << mc_obj->mc_get_error(result) << endl;
   }
   return result;
}

//===================================================================
// This function will receive data from the supplied connection and
// break it into three mshn_data types.
int   recv_3_data(
                  mshn_com     *mc_obj,
                  mshn_data    *dt1,
                  char         *dt1_label,
                  mshn_data    *dt2,
                  char         *dt2_label,
                  mshn_data    *dt3,
```

```
                    char          *dt3_label,
                    int           &bytes_rec)
{
   int result = MSHN_COM_OK;
   // Clean out the outputs
   dt1->the_data = NULL;
   dt2->the_data = NULL;
   dt3->the_data = NULL;
   dt1->the_length = 0;
   dt2->the_length = 0;
   dt3->the_length = 0;
   mshn_data new_buff;

   new_buff.the_data = (unsigned char *)
                        mshn_malloc(BUFF_SIZE * 16, NULL);
   new_buff.the_length = BUFF_SIZE * 16;
   result = mc_obj->mc_recv(new_buff.the_data, new_buff.the_length,
            bytes_rec);

   unsigned char *new_buff_ptr = new_buff.the_data;
   get_mshn_data(&new_buff_ptr, dt1);
   get_mshn_data(&new_buff_ptr, dt2);
   get_mshn_data(&new_buff_ptr, dt3);

   if (result == MSHN_COM_OK) {
      cout << "Received (" << bytes_rec << ") bytes." << endl;
      cout << dt1_label << " length (" << dt1->the_length << ") bytes."
           << endl;
      cout << dt2_label << " length (" << dt2->the_length << ") bytes."
           << endl;
      cout << dt3_label << " length (" << dt3->the_length << ") bytes."
           << endl;
   }else{
      cout << "mc_recv: " << mc_obj->mc_get_error(result) << endl;
   }
   return result;
}

//=================================================================
// This function will bundle up an integer and 3 mshn_data types and
// send them out the supplied connection.
int   send_int_3_data(
                    mshn_com          *mc_obj,
                    int                it1,
                    const char        *it1_label,
                    const mshn_data   *dt1,
                    const char        *dt1_label,
                    const mshn_data   *dt2,
                    const char        *dt2_label,
                    const mshn_data   *dt3,
                    const char        *dt3_label,
```

```
                    int                &bytes_sent)
{
   int result = MSHN_COM_OK;
   int total_length =  sizeof(int) + dt1->the_length + dt2->the_length
                      + dt3->the_length + sizeof(unsigned int) * 3;
   cout << "total length " << total_length << endl;
   mshn_data new_buff;
   new_buff.the_length = total_length;
   new_buff.the_data = (unsigned char *)mshn_malloc(total_length, NULL);
   unsigned char *new_buff_ptr = new_buff.the_data;
   add_int(&new_buff_ptr, it1);
   add_mshn_data(&new_buff_ptr, dt1);
   add_mshn_data(&new_buff_ptr, dt2);
   add_mshn_data(&new_buff_ptr, dt3);

   cout << "Sending everything: " << new_buff.the_length << " bytes."
        << endl;
   result = mc_obj->mc_send(new_buff.the_data, new_buff.the_length,
            bytes_sent);

   if (result == MSHN_COM_OK) {
      cout << "Sent (" << bytes_sent << ") bytes." << endl;
      cout << it1_label << "        (" << it1 << ")" << endl;
      cout << dt1_label << " length (" << dt1->the_length << ") bytes."
           << endl;
      cout << dt2_label << " length (" << dt2->the_length << ") bytes."
           << endl;
      cout << dt3_label << " length (" << dt3->the_length << ") bytes."
           << endl;
   }else{
      cout << "mc_send: " << mc_obj->mc_get_error(result) << endl;
   }
   mshn_free(new_buff.the_data, NULL);
   return result;
}

//=================================================================
// This function will receive data from the supplied connection and
// break it into an integer and 3 mshn_data types.
int   recv_int_3_data(
                  mshn_com     *mc_obj,
                  int          *it1,
                  char         *it1_label,
                  mshn_data    *dt1,
                  char         *dt1_label,
                  mshn_data    *dt2,
                  char         *dt2_label,
                  mshn_data    *dt3,
                  char         *dt3_label,
                  int          &bytes_rec)
{
```

```
    int result = MSHN_COM_OK;
    // Clean out the outputs
    *it1 = -1;
    dt1->the_data = NULL;
    dt2->the_data = NULL;
    dt3->the_data = NULL;
    dt1->the_length = 0;
    dt2->the_length = 0;
    dt3->the_length = 0;
    mshn_data new_buff;

    new_buff.the_data = (unsigned char *)
                        mshn_malloc(BUFF_SIZE * 16, NULL);
    new_buff.the_length = BUFF_SIZE * 16;
    result = mc_obj->mc_recv(new_buff.the_data, new_buff.the_length,
             bytes_rec);

    unsigned char *new_buff_ptr = new_buff.the_data;
    get_int(&new_buff_ptr, it1);
    get_mshn_data(&new_buff_ptr, dt1);
    get_mshn_data(&new_buff_ptr, dt2);
    get_mshn_data(&new_buff_ptr, dt3);

    if (result == MSHN_COM_OK) {
       cout << "Received (" << bytes_rec << ") bytes." << endl;
       cout << it1_label << "        (" << *it1 << ")" << endl;
       cout << dt1_label << " length (" << dt1->the_length << ") bytes."
            << endl;
       cout << dt2_label << " length (" << dt2->the_length << ") bytes."
            << endl;
       cout << dt3_label << " length (" << dt3->the_length << ") bytes."
            << endl;
    }else{
       cout << "mc_recv: " << mc_obj->mc_get_error(result) << endl;
    }
    return result;
}

//=====================================================================
// This function will bundle up an integer and 4 mshn_data types and
// send them out the supplied connection.
int   send_int_4_data(
                  mshn_com          *mc_obj,
                  int                it1,
                  const char        *it1_label,
                  const mshn_data   *dt1,
                  const char        *dt1_label,
                  const mshn_data   *dt2,
                  const char        *dt2_label,
                  const mshn_data   *dt3,
                  const char        *dt3_label,
```

```
                    const mshn_data    *dt4,
                    const char         *dt4_label,
                    int                &bytes_sent)
{
   int result = MSHN_COM_OK;
   int total_length = sizeof(int) + dt1->the_length + dt2->the_length
         + dt3->the_length + dt4->the_length + sizeof(unsigned int) * 4;
   cout << "total length " << total_length << endl;
   mshn_data new_buff;
   new_buff.the_length = total_length;
   new_buff.the_data = (unsigned char *)
                        mshn_malloc(total_length, NULL);
   unsigned char *new_buff_ptr = new_buff.the_data;
   add_int(&new_buff_ptr, it1);
   add_mshn_data(&new_buff_ptr, dt1);
   add_mshn_data(&new_buff_ptr, dt2);
   add_mshn_data(&new_buff_ptr, dt3);
   add_mshn_data(&new_buff_ptr, dt4);

   cout << "Sending everything: " << new_buff.the_length << " bytes."
        << endl;
   result = mc_obj->mc_send(new_buff.the_data, new_buff.the_length,
         bytes_sent);

   if (result == MSHN_COM_OK) {
      cout << "Sent (" << bytes_sent << ") bytes." << endl;
      cout << it1_label << "         (" << it1 << ")" << endl;
      cout << dt1_label << " length (" << dt1->the_length << ") bytes."
           << endl;
      cout << dt2_label << " length (" << dt2->the_length << ") bytes."
           << endl;
      cout << dt3_label << " length (" << dt3->the_length << ") bytes."
           << endl;
      cout << dt4_label << " length (" << dt4->the_length << ") bytes."
           << endl;
   }else{
      cout << "mc_send: " << mc_obj->mc_get_error(result) << endl;
   }
   mshn_free(new_buff.the_data, NULL);
   return result;
}

//======================================================================
// This function will receive data from the supplied connection and
// break it into an integer and 4 mshn_data types.
int   recv_int_4_data(
                    mshn_com     *mc_obj,
                    int          *it1,
                    char         *it1_label,
                    mshn_data    *dt1,
                    char         *dt1_label,
```

```
                    mshn_data     *dt2,
                    char          *dt2_label,
                    mshn_data     *dt3,
                    char          *dt3_label,
                    mshn_data     *dt4,
                    char          *dt4_label,
                    int           &bytes_rec)
{
   int result = MSHN_COM_OK;
   // Clean out the outputs
   *it1 = -1;
   dt1->the_data = NULL;
   dt2->the_data = NULL;
   dt3->the_data = NULL;
   dt4->the_data = NULL;
   dt1->the_length = 0;
   dt2->the_length = 0;
   dt3->the_length = 0;
   dt4->the_length = 0;
   mshn_data new_buff;

   new_buff.the_data = (unsigned char *)
                         mshn_malloc(BUFF_SIZE * 16, NULL);
   new_buff.the_length = BUFF_SIZE * 16;
   result = mc_obj->mc_recv(new_buff.the_data, new_buff.the_length,
            bytes_rec);

   unsigned char *new_buff_ptr = new_buff.the_data;
   get_int(&new_buff_ptr, it1);
   get_mshn_data(&new_buff_ptr, dt1);
   get_mshn_data(&new_buff_ptr, dt2);
   get_mshn_data(&new_buff_ptr, dt3);
   get_mshn_data(&new_buff_ptr, dt4);

   if (result == MSHN_COM_OK) {
      cout << "Received (" << bytes_rec << ") bytes." << endl;
      cout << it1_label << "         (" << *it1 << ")" << endl;
      cout << dt1_label << " length (" << dt1->the_length << ") bytes."
           << endl;
      cout << dt2_label << " length (" << dt2->the_length << ") bytes."
           << endl;
      cout << dt3_label << " length (" << dt3->the_length << ") bytes."
           << endl;
      cout << dt4_label << " length (" << dt4->the_length << ") bytes."
           << endl;
   }else{
      cout << "mc_recv: " << mc_obj->mc_get_error(result) << endl;
   }
   return result;
}
```

```
//===========================================================================
// This function will bundle up 6 mshn_data types and send
// them out the supplied connection.
int   send_6_data(
                  mshn_com          *mc_obj,
                  const mshn_data   *dt1,
                  const char        *dt1_label,
                  const mshn_data   *dt2,
                  const char        *dt2_label,
                  const mshn_data   *dt3,
                  const char        *dt3_label,
                  const mshn_data   *dt4,
                  const char        *dt4_label,
                  const mshn_data   *dt5,
                  const char        *dt5_label,
                  const mshn_data   *dt6,
                  const char        *dt6_label,
                  int               &bytes_sent)
{
   int result = MSHN_COM_OK;
   int total_length = dt1->the_length + dt2->the_length +
      dt3->the_length + dt4->the_length + dt5->the_length +
      dt6->the_length + sizeof(unsigned int) * 6;
   cout << "total length " << total_length << endl;
   mshn_data new_buff;
   new_buff.the_length = total_length;
   new_buff.the_data = (unsigned char *)mshn_malloc(total_length, NULL);
   unsigned char *new_buff_ptr = new_buff.the_data;
   add_mshn_data(&new_buff_ptr, dt1);
   add_mshn_data(&new_buff_ptr, dt2);
   add_mshn_data(&new_buff_ptr, dt3);
   add_mshn_data(&new_buff_ptr, dt4);
   add_mshn_data(&new_buff_ptr, dt5);
   add_mshn_data(&new_buff_ptr, dt6);

   cout << "Sending everything: " << new_buff.the_length << " bytes."
        << endl;
   result = mc_obj->mc_send(new_buff.the_data, new_buff.the_length,
            bytes_sent);

   if (result == MSHN_COM_OK) {
      cout << "Sent (" << bytes_sent << ") bytes." << endl;
      cout << dt1_label << " length (" << dt1->the_length << ") bytes."
           << endl;
      cout << dt2_label << " length (" << dt2->the_length << ") bytes."
           << endl;
      cout << dt3_label << " length (" << dt3->the_length << ") bytes."
           << endl;
      cout << dt4_label << " length (" << dt4->the_length << ") bytes."
           << endl;
      cout << dt5_label << " length (" << dt5->the_length << ") bytes."
```

```
            << endl;
        cout << dt6_label << " length (" << dt6->the_length << ") bytes."
            << endl;
    }else{
        cout << "mc_send: " << mc_obj->mc_get_error(result) << endl;
    }
    mshn_free(new_buff.the_data, NULL);
    return result;
}

//===================================================================
// This function will receive data from the supplied connection and
// break it into 6 mshn_data types.
int   recv_6_data(
                  mshn_com      *mc_obj,
                  mshn_data     *dt1,
                  char          *dt1_label,
                  mshn_data     *dt2,
                  char          *dt2_label,
                  mshn_data     *dt3,
                  char          *dt3_label,
                  mshn_data     *dt4,
                  char          *dt4_label,
                  mshn_data     *dt5,
                  char          *dt5_label,
                  mshn_data     *dt6,
                  char          *dt6_label,
                  int           &bytes_rec)
{
    int result = MSHN_COM_OK;
    // Clean out the outputs
    dt1->the_data = NULL;
    dt2->the_data = NULL;
    dt3->the_data = NULL;
    dt4->the_data = NULL;
    dt5->the_data = NULL;
    dt6->the_data = NULL;
    dt1->the_length = 0;
    dt2->the_length = 0;
    dt3->the_length = 0;
    dt4->the_length = 0;
    dt5->the_length = 0;
    dt6->the_length = 0;
    mshn_data new_buff;

    new_buff.the_data = (unsigned char *)
                          mshn_malloc(BUFF_SIZE * 16, NULL);
    new_buff.the_length = BUFF_SIZE * 16;
    result = mc_obj->mc_recv(new_buff.the_data, new_buff.the_length,
              bytes_rec);
```

```
   unsigned char *new_buff_ptr = new_buff.the_data;
   get_mshn_data(&new_buff_ptr, dt1);
   get_mshn_data(&new_buff_ptr, dt2);
   get_mshn_data(&new_buff_ptr, dt3);
   get_mshn_data(&new_buff_ptr, dt4);
   get_mshn_data(&new_buff_ptr, dt5);
   get_mshn_data(&new_buff_ptr, dt6);

   if (result == MSHN_COM_OK) {
      cout << "Received (" << bytes_rec << ") bytes." << endl;
      cout << dt1_label << " length (" << dt1->the_length << ") bytes."
           << endl;
      cout << dt2_label << " length (" << dt2->the_length << ") bytes."
           << endl;
      cout << dt3_label << " length (" << dt3->the_length << ") bytes."
           << endl;
      cout << dt4_label << " length (" << dt4->the_length << ") bytes."
           << endl;
      cout << dt5_label << " length (" << dt5->the_length << ") bytes."
           << endl;
      cout << dt6_label << " length (" << dt6->the_length << ") bytes."
           << endl;
   }else{
      cout << "mc_recv: " << mc_obj->mc_get_error(result) << endl;
   }
   return result;
}

//=====================================================================
// Convert from a mshn_token structure to a mshn_data structure
void  token_to_mshn_data (
                  const mshn_token   the_token,
                  mshn_data          *the_data)
{
   the_data->the_length = sizeof(the_token.request_id) +
      the_token.job_session_key.the_length;
   the_data->the_data = (unsigned char *)
      mshn_malloc(the_data->the_length, NULL);
   memcpy(the_data->the_data, &(the_token.request_id),
      sizeof(the_token.request_id));
   memcpy(the_data->the_data + sizeof(the_token.request_id),
      the_token.job_session_key.the_data,
      the_token.job_session_key.the_length);
}

//=====================================================================
// Convert from a mshn_data structure to a mshn_token structure
void  mshn_data_to_token (
            const mshn_data  *the_data,
                  mshn_token *the_token)
{
```

```
    memcpy(&(the_token->request_id), the_data->the_data,
        sizeof(the_token->request_id));
    the_token->job_session_key.the_length = the_data->the_length -
        sizeof(the_token->request_id);
    the_token->job_session_key.the_data = (unsigned char *)
        mshn_malloc(the_token->job_session_key.the_length, NULL);
    memcpy(the_token->job_session_key.the_data, the_data->the_data +
        sizeof(the_token->request_id),
        the_token->job_session_key.the_length);
}


//======================================================================
//  obtains the user ID, certificate, passphrase, communications
//  security option and certificate validation level from the user
//
int do_register(mshn_sl *msl_obj,
                mshn_data &user_id,
                mshn_data &user_cert,
                char **passphrase,
                comm_security &com_sec_option,
                cert_checking &cert_valid_level)
{
   int result = 0;
   char response[80];
   char *err_out;

   // Clear the screen, so that getpass prompts correctly
   clrscr();

   cout << endl << "Enter your certificate name: ";
   cin.getline(response, 80);

   result = msl_obj->mshn_sl_get_cert(response, &user_cert);

   if (result != MSHN_OK) {
      err_out = msl_obj->mshn_sl_show_error(result);
      cout << "mshn_sl_get_certificate " << err_out << endl;
      mshn_free(err_out, NULL);
   }
   else {
      cout << "Certificate ok" << endl << endl;
      user_id.the_length = strlen(response);
      user_id.the_data = (unsigned char *)
         mshn_malloc(user_id.the_length, NULL);
      memcpy(user_id.the_data, response, user_id.the_length);

      *passphrase = getpass("Enter the pass phrase
                             for your private key: ");

      do {
```

```
        cout << endl << "Enter the communications security option: "
             << endl;
        cout          << "   0 - None" << endl;
        cout          << "   1 - Integrity" << endl;
        cout          << "   2 - Confidentiality" << endl;
        cout          << "   3 - Both" << endl;

        cin.getline(response, 80);
      }
        while (atoi(response) < 0 || atoi(response) > 3);

      com_sec_option = atoi(response);

      do {
          cout << endl << "Enter the certificate validation level: "
               << endl;
          cout          << "   0 - None" << endl;
          cout          << "   1 - Check Authenticity" << endl;
          cout          << "   2 - Check Revocation List" << endl;
          cout          << "   3 - Both" << endl;

        cin.getline(response, 80);
        }
      while (atoi(response) < 0 || atoi(response) > 3);

      cert_valid_level = atoi(response);

   }
      return result;
}

//=======================================================================
// checks the given certificate for authenticity
int do_cert_check(mshn_sl            *the_sl,
                  cert_checking      c_check,
                  mshn_data          *the_cert)
{
   int result = MSHN_OK;
   char *err_out;

   int cert_valid, cert_revoked;
   if ((c_check == CERT_AUTH) || (c_check == CERT_BOTH)) {
      result = the_sl->mshn_sl_cert_verify(the_cert, &cert_valid);
      if (result != MSHN_OK) {
         err_out = the_sl->mshn_sl_show_error(result);
         cout << "mshn_sl_cert_verify " << err_out << endl;
         mshn_free(err_out, NULL);
      }else{
         if (cert_valid == MSHN_FALSE) {
            result = MSHN_CERT_INVALID;
            err_out = the_sl->mshn_sl_show_error(result);
```

```
            cout << "mshn_sl_cert_verify " << err_out << endl;
            mshn_free(err_out, NULL);
         }
      }
   }

   if (result == MSHN_OK) {
      if ((c_check == CERT_REV) || (c_check == CERT_BOTH)) {
            result = the_sl->mshn_sl_cert_revoked(the_cert,
                     &cert_revoked);
            if (result != MSHN_OK) {
                  err_out = the_sl->mshn_sl_show_error(result);
            cout << "mshn_sl_cert_revoked " << err_out << endl;
            mshn_free(err_out, NULL);
            }else{
               if (cert_revoked == MSHN_TRUE) {
                  result = MSHN_CERT_REVOKED;
                  err_out = the_sl->mshn_sl_show_error(result);
                  cout << "mshn_sl_cert_revoked " << err_out << endl;
                  mshn_free(err_out, NULL);
               }
         }
      }
   }
   return result;
}
```

J. MSHNUTIL.H

```
// MSHN Utility Function Header File
// Written by David Shifflet & Roger Wright
//
#ifndef _MSHNUTIL_H
#define _MSHNUTIL_H

void MSHN_CSP_INIT(CSSM_LIST_PTR pGUIDList,
                   CSSM_MODULE_INFO_PTR pInfo,
                   CSSM_CSP_HANDLE &hCSP);

void MSHN_CL_INIT (CSSM_LIST_PTR pGUIDList,
                   CSSM_MODULE_INFO_PTR pInfo,
                   CSSM_CL_HANDLE &hCL);

void MSHN_DL_INIT (CSSM_LIST_PTR pGUIDList,
                   CSSM_MODULE_INFO_PTR pInfo,
                   CSSM_DL_HANDLE &hDL,
                   CSSM_DB_HANDLE &hDS,
                   char *ds_names[]);
```

```
CSSM_RETURN MSHN_CSSM_INIT();

CSSM_KEY_PTR MSHN_GenerateKey(CSSM_CSP_HANDLE hCSP,
                                 uint32 keySizeInBits,
                                 const char* passPhrase);

CSSM_DATA_PTR MSHN_EncryptData(CSSM_CSP_HANDLE hCSP,
                              CSSM_DATA_PTR pClear,    // bits to encrypt
                              CSSM_KEY_PTR pKey);

CSSM_DATA_PTR MSHN_DecryptData(CSSM_CSP_HANDLE hCSP,
                        CSSM_DATA_PTR pEncrypted, // bits to decrypt
                        CSSM_KEY_PTR pKey);

//////////////////////////////////////////////////////////////////////////
CSSM_RETURN MSHN_GenerateKeyPair(CSSM_CSP_HANDLE hCSP,
                                 CSSM_KEY_PTR publicKey,
                                 CSSM_KEY_PTR privateKey,
                                 char * ppassphrase);

//////////////////////////////////////////////////////////////////////////
CSSM_DATA_PTR MSHN_SignData(CSSM_CSP_HANDLE hCSP,
                            CSSM_DATA_PTR pClear,      // bits to sign
                            CSSM_KEY_PTR pKey,
                            char * password);

//////////////////////////////////////////////////////////////////////////
CSSM_BOOL MSHN_VerifyData(CSSM_CSP_HANDLE hCSP,
                          CSSM_DATA_PTR pSigned,       // bits to verify
                          CSSM_DATA_PTR pClear,
                          CSSM_KEY_PTR pKey);

//////////////////////////////////////////////////////////////////////////
CSSM_DATA_PTR MSHN_DigestData(CSSM_CSP_HANDLE hCSP,
                              CSSM_DATA_PTR pClearText);

//////////////////////////////////////////////////////////////////////////
CSSM_KEY_PTR MSHN_GenerateDESKey(CSSM_CSP_HANDLE hCSP,
                                 uint32 keySizeInBits);

//////////////////////////////////////////////////////////////////////////
CSSM_WRAP_KEY_PTR MSHN_WrapKey(CSSM_CSP_HANDLE hCSP,
                               CSSM_KEY_PTR psymmetricKey,
                               CSSM_KEY_PTR pmasterKey);

//////////////////////////////////////////////////////////////////////////
CSSM_KEY_PTR MSHN_UnwrapKey(CSSM_CSP_HANDLE hCSP,
                            CSSM_WRAP_KEY_PTR psymmetricKey,
                            CSSM_KEY_PTR pmasterKey);

#endif
```

K. MSHNUTIL.CPP

```
// MSHN Utility Function Definitions
// Written by Roger Wright & David Shifflett
//

#include <iostream.h>
#include <stdlib.h>
#include <fstream.h>
#include <mem.h>
#include <string.h>
#include <cssm.h>
#include "mshnUtil.h"
#include "mshn_mem.h"
#include "showutil.h"


#define FALSE 0

//cssm memory functions
void * mshn_malloc (uint32 size, void *allocRef)
{
   return(malloc(size));
}

void  mshn_free (void *mem_ptr, void *allocRef)
{
   if (mem_ptr != NULL) {
      free(mem_ptr);
   }
   return;
}

void * mshn_realloc (void *ptr, uint32 size, void *allocRef)
{
   return(realloc(ptr, size));
}

void * mshn_calloc (uint32 num, uint32 size, void *allocRef)
{
   return(calloc(num, size));
}


CSSM_API_MEMORY_FUNCS mem_fx = {mshn_malloc,
                                mshn_free,
                                mshn_realloc,
                                mshn_calloc,
                                NULL};
```

```
//////////////////////////////////////////////////////////////////////////
// initialize the CSSM
CSSM_RETURN MSHN_CSSM_INIT()
{
   CSSM_RETURN ret;
   CSSM_VERSION version;
   version.Major = CSSM_MAJOR;
   version.Minor = CSSM_MINOR;
   ret = CSSM_Init (&version, &mem_fx, NULL);

   return(ret);
}

//////////////////////////////////////////////////////////////////////////
// MSHN CSP Initialization Function
//
void MSHN_CSP_INIT(CSSM_LIST_PTR pGUIDList,
                   CSSM_MODULE_INFO_PTR pInfo,
                   CSSM_CSP_HANDLE &hCSP)
{
   hCSP = CSSM_INVALID_HANDLE;
   pGUIDList = CSSM_ListModules(CSSM_SERVICE_CSP, FALSE);
      if (!pGUIDList)
   {
      cout << "Error listing CSP modules\n";
   }else{
      // using the first module
      pInfo = CSSM_GetModuleInfo(&(pGUIDList->Items[0].GUID),
                                 CSSM_SERVICE_CSP,
                                 CSSM_ALL_SUBSERVICES,
                                 CSSM_INFO_LEVEL_SUBSERVICE);

      if (!pInfo)
      {
            cout << "Error getting CSP module info.\n";
      }else{
            hCSP = CSSM_ModuleAttach(&pGUIDList->Items[0].GUID,
                                     &pInfo->Version,
                                     &mem_fx,
                                     0,     // subservice id
                                     CSSM_SERVICE_CSP,
                                     0,
                                     0,
                                     0);

            if (hCSP==NULL) {
            show_error("CSP CSSM_ModuleAttach");
            cout << "CSP Did not Initialize OK " << endl;
            }

            // free module info memory
```

```
            CSSM_FreeModuleInfo(pInfo);
        }
        CSSM_FreeList(pGUIDList);
    }

    return;
}

////////////////////////////////////////////////////////////////////////
// MSHN CL Initialization Function
//
void MSHN_CL_INIT(CSSM_LIST_PTR pGUIDList,
                  CSSM_MODULE_INFO_PTR pInfo,
                  CSSM_CL_HANDLE &hCL)
{
    hCL = CSSM_INVALID_HANDLE;
    pGUIDList = CSSM_ListModules(CSSM_SERVICE_CL, FALSE);
    if (!pGUIDList)
    {
        cout << "Error listing CL modules\n";
    }else{
        // using the first module
        pInfo = CSSM_GetModuleInfo(&(pGUIDList->Items[0].GUID),
                                   CSSM_SERVICE_CL,
                                   CSSM_ALL_SUBSERVICES,
                                   CSSM_INFO_LEVEL_ALL_ATTR);

        if (!pInfo)
        {
            cout << "Error getting CL module info\n";
        }else{
            hCL = CSSM_ModuleAttach(&pGUIDList->Items[0].GUID,
                                    &pInfo->Version,
                                    &mem_fx,
                                    0,     // subservice id
                                    CSSM_SERVICE_CL,
                                    0,
                                    0,
                                    0);
            if (hCL==NULL) {
            show_error("CL CSSM_ModuleAttach");
            cout << "CL Did not Initialize OK " << endl;
            }

            // free module info memory
            CSSM_FreeModuleInfo(pInfo);
        }
        CSSM_FreeList(pGUIDList);
    }
    return;
}
```

```
////////////////////////////////////////////////////////////////////////
// MSHN DL Initialization Function
//
void MSHN_DL_INIT(CSSM_LIST_PTR pGUIDList,
                   CSSM_MODULE_INFO_PTR pInfo,
                   CSSM_DL_HANDLE &hDL,
                   CSSM_DB_HANDLE &hDS,
                   char *ds_names[])
{
   hDL = CSSM_INVALID_HANDLE;
   pGUIDList = CSSM_ListModules(CSSM_SERVICE_DL, FALSE);
   if (!pGUIDList)
   {
      cout << "Error listing DL modules\n";
   }else{
      // using the first module
      pInfo = CSSM_GetModuleInfo(&(pGUIDList->Items[0].GUID),
                                 CSSM_SERVICE_DL,
                                 CSSM_ALL_SUBSERVICES,
                                 CSSM_INFO_LEVEL_ALL_ATTR);

      if (!pInfo)
      {
            cout << "Error getting module info.\n";
      }else{
            hDL = CSSM_ModuleAttach(&pGUIDList->Items[0].GUID,
                                    &pInfo->Version,
                                    &mem_fx,
                                    0,     // subservice id
                                    CSSM_SERVICE_DL,
                                    0,
                                    0,
                                    0);

            if (hDL==NULL) {
            show_error("DL CSSM_ModuleAttach");
            cout << "DL Did not Initialize OK " << endl;
            }else{
                  // now display the data store names
            char dbname[80];

            // We should really have some sort of loop to handle all the
            // passed in names, and output all the handles
            if (ds_names[0] == NULL) {
               cout << endl << "Available data store names" << endl;
               show_datastore_names((CSSM_DLSUBSERVICE_PTR)
                                   pInfo->ServiceList->SubServiceList);
               cout << endl << "enter one of the above names " << endl;
               cin.getline(dbname, 80);
            }else{
```

```
                strcpy(dbname, ds_names[0]);
          }
                // Now try and open the data store
                CSSM_DB_ACCESS_TYPE dbaccess;
                dbaccess.ReadAccess = CSSM_TRUE;
                dbaccess.WriteAccess = CSSM_TRUE;
                dbaccess.PrivilegedMode = CSSM_FALSE;
                dbaccess.Asynchronous = CSSM_FALSE;
                CSSM_DB_HANDLE db_hand = CSSM_DL_DbOpen(hDL,
                   dbname, &dbaccess, NULL, NULL);
                if (db_hand == CSSM_INVALID_HANDLE) {
                   show_error("DL CSSM_DL_DbOpen");
                   cout << "Failed to attach data store" << endl;
          }else{
                hDS = db_hand;
               }
          }

          // free module info memory
          CSSM_FreeModuleInfo(pInfo);
      }
      CSSM_FreeList(pGUIDList);
   }
   return;
}

////////////////////////////////////////////////////////////////////////
// Generates a DES symmetric key based on the given passphrase
//
CSSM_KEY_PTR MSHN_GenerateKey(CSSM_CSP_HANDLE hCSP,
                              uint32 keySizeInBits,
                              const char* passPhrase)
{
      CSSM_RETURN cssmstatus;
      CSSM_CC_HANDLE hCC = NULL;
      CSSM_DATA cssmPassPhrase;

      // copy passPhrase to get rid of const

      char * pLocalPassPhrase = NULL;
      pLocalPassPhrase = (char*)mem_fx.malloc_func
           (strlen(passPhrase) +1,NULL);
      strcpy(pLocalPassPhrase,passPhrase);

      // CSSMize it
      cssmPassPhrase.Length = strlen(pLocalPassPhrase);
      cssmPassPhrase.Data = (unsigned char*) pLocalPassPhrase;

      CSSM_CRYPTO_DATA passPhraseData;
      passPhraseData.CallbackID = 0;
      passPhraseData.Callback = NULL;
```

```
    passPhraseData.Param = &cssmPassPhrase;

    hCC = CSSM_CSP_CreateDeriveKeyContext(hCSP,        // CSP handle
                                CSSM_ALGID_MD5_PBE, // alg ID
                                CSSM_ALGID_DES,     // key type
                                keySizeInBits,      // key size
                                0,                  // iteration count
                                NULL,               // salt
                                NULL,               // seed
                                &passPhraseData);   // passphrase


    if(hCC == NULL)
    {
          cout << "Error creating key generation context" << endl;
          CSSM_ModuleDetach (hCSP);
          return NULL;
    }
    CSSM_KEY_PTR pKey =
    (CSSM_KEY_PTR)mem_fx.malloc_func(sizeof(CSSM_KEY),NULL);

    // setting these fields to NULL will tell the CSP to allocate the
    // memory for us
    pKey->KeyData.Data =  NULL;
    pKey->KeyData.Length = 0;

    cssmstatus = CSSM_DeriveKey(hCC,     // context handle
                         NULL, // base key
                         NULL, // params
                         CSSM_KEYUSE_ANY, // usage
                         CSSM_KEYATTR_RETURN_DEFAULT, // attributes
                         NULL, // label
                         pKey ); // the key


    CSSM_DeleteContext(hCC);

    if(cssmstatus != CSSM_OK)
    {
          cout << "Error creating CSSM key" << endl;
          return NULL;
    }
    return pKey;
}

//////////////////////////////////////////////////////////////////////////
CSSM_DATA_PTR MSHN_EncryptData(CSSM_CSP_HANDLE hCSP,
                               CSSM_DATA_PTR pClear,     // bits to encrypt
                               CSSM_KEY_PTR pKey)

{

    CSSM_RETURN cssmstatus;
```

```
CSSM_CC_HANDLE hCC;
CSSM_DATA remData;
remData.Length = 0;
remData.Data = 0;

// this is the return value
CSSM_DATA_PTR pEncrypted =
(CSSM_DATA_PTR)mem_fx.malloc_func(sizeof(CSSM_DATA),NULL);
pEncrypted->Data = NULL;
pEncrypted->Length = 0;

hCC = CSSM_CSP_CreateSymmetricContext(hCSP,
                                    CSSM_ALGID_DES,
                                    CSSM_ALGMODE_CBCPadIV8,
                                    pKey,
                                    NULL,    // no initial vector
                                    CSSM_PADDING_PKCS5,
                                    0        // 0 rounds
                                    );
if (hCC == 0)
{
      cout << "Error creating decrypt context" << endl;
      return NULL;

}

CSSM_QUERY_SIZE_DATA queryData;

queryData.SizeInputBlock = pClear->Length;
queryData.SizeOutputBlock = 0;

CSSM_QuerySize(hCC,
                        CSSM_TRUE,  // for encryption
                        1,                // 1 block
                        &queryData);

// allocate memory to hold the encrypted bits
pEncrypted->Length = queryData.SizeOutputBlock;
pEncrypted->Data = (unsigned char*)
      mem_fx.malloc_func(queryData.SizeOutputBlock,NULL);

unsigned int bytesEncrypted;
cssmstatus = CSSM_EncryptData(hCC,
                              pClear,
                              1,
                              pEncrypted,
                              1,
                              &bytesEncrypted,
                              &remData
                              );
```

```
    if(remData.Data != NULL)
    {
        mem_fx.free_func(remData.Data,NULL);
    }
    CSSM_DeleteContext(hCC);

    if (cssmstatus != CSSM_OK)
    {
        return NULL;
    }
    else
    {
        return pEncrypted;
    }
}

/////////////////////////////////////////////////////////////////////////////
CSSM_DATA_PTR MSHN_DecryptData(CSSM_CSP_HANDLE hCSP,
                CSSM_DATA_PTR pEncrypted,     // bytes to decrypt
                CSSM_KEY_PTR pKey)
{
    CSSM_RETURN cssmstatus;
    CSSM_CC_HANDLE hCC;
    CSSM_DATA remData;
    remData.Length = 0;
    remData.Data = 0;

    // this is the return value
    CSSM_DATA_PTR pClear =
    (CSSM_DATA_PTR)mem_fx.malloc_func(sizeof(CSSM_DATA),NULL);
    pClear->Data = NULL;
    pClear->Length = 0;

    hCC = CSSM_CSP_CreateSymmetricContext(hCSP,
                        CSSM_ALGID_DES,
                        CSSM_ALGMODE_CBCPadIV8,
                        pKey,
                        NULL,   // no initial vector
                        CSSM_PADDING_PKCS5,     //padding
                        0       // 0 rounds
                        );

    if (hCC == 0)
    {
        cout << "Error creating decrypt context" << endl;
        return NULL;
    }

    CSSM_QUERY_SIZE_DATA queryData;

    queryData.SizeInputBlock = pEncrypted->Length;
```

```
    queryData.SizeOutputBlock = 0;

    CSSM_QuerySize(hCC,
                CSSM_FALSE,  // not for encryption
                1,                // 1 block
                &queryData);

    // allocate memory to hold the decrypted bits
    pClear->Length = queryData.SizeOutputBlock;
    pClear->Data = (unsigned char*)
          mem_fx.malloc_func(queryData.SizeOutputBlock,NULL);

    if(pClear->Data == NULL)
    {
          cout << "Throw Memory Exception" << endl;
    }

    unsigned int bytesDecrypted;

    cssmstatus = CSSM_DecryptData(hCC,
                                  pEncrypted,
                                  1,
                                  pClear,
                                  1,
                                  &bytesDecrypted,
                                  &remData);

    if(remData.Data != NULL)
    {
          mem_fx.free_func(remData.Data,NULL);
    }

    CSSM_DeleteContext(hCC);

    if (cssmstatus != CSSM_OK)
    {
          cout << "Error decrypting data" << endl;
          return NULL;
    }
    else
    {
          return pClear;
    }
}

////////////////////////////////////////////////////////////////////////
CSSM_RETURN MSHN_GenerateKeyPair(CSSM_CSP_HANDLE hCSP,
                                 CSSM_KEY_PTR publicKey,
                                 CSSM_KEY_PTR privateKey,
                                 char * ppassphrase)
{
```

```
CSSM_RETURN result = CSSM_OK;
CSSM_CC_HANDLE keyGenContext = NULL;

CSSM_CRYPTO_DATA            seed;
CSSM_DATA                   seedData;
CSSM_CRYPTO_DATA            password;
CSSM_DATA                   password_data;

int passphraseLength = 0;

passphraseLength = strlen(ppassphrase);
cout << "pass phrase length: " << passphraseLength << endl;
// find length of pass phrase

// Create KPG context.  Use password as the key generation seed
seedData.Length = passphraseLength;   // default passphrase length
seedData.Data = (unsigned char*)
    mem_fx.malloc_func(seedData.Length,NULL);
memcpy(seedData.Data,ppassphrase,seedData.Length);

seed.Param = &seedData;
seed.Callback = NULL;

// Initialize the password information
password_data.Length = passphraseLength;
password_data.Data = (unsigned char*)
   mem_fx.malloc_func(password_data.Length,NULL);
memcpy(password_data.Data,ppassphrase,password_data.Length);

password.Param = &password_data;
password.Callback = NULL;

cout << "passphrase : ";
for (int t = 0; t < passphraseLength; t++) {
   cout << password_data.Data[t];
}
cout << "*" << endl;

keyGenContext = CSSM_CSP_CreateKeyGenContext(hCSP,
                     CSSM_ALGID_DSA,    // alg ID
                     &password,         // password
                     512,               // key size in bits
                     &seed,             // seed
                     NULL,              // salt
                     NULL,              // start date
                     NULL,              // end date
                     NULL);             // optional params

if (!keyGenContext)
   {
        cout << "Failed to create a key generation context" << endl;
```

```
            return CSSM_FAIL;
      }

      // tell CSSM to allocate memory for key
      publicKey->KeyData.Length = 0;
      publicKey->KeyData.Data = NULL;

      privateKey->KeyData.Length = 0;
      privateKey->KeyData.Data = NULL;

      result = CSSM_GenerateKeyPair(keyGenContext,
                        CSSM_KEYUSE_ANY, // usage
                        CSSM_KEYATTR_RETURN_DEFAULT, // attributes
                        NULL, // label
                        publicKey,
                        CSSM_KEYUSE_ANY, // usage
                        CSSM_KEYATTR_RETURN_DEFAULT, // attributes
                        NULL, // label
                        privateKey);

      CSSM_DeleteContext(keyGenContext);

   if(result != CSSM_OK)
      {
            cout << "Failed to generate key" << endl;
      }

   cout << "public key : " << publicKey->KeyData.Data << endl;
   return result;
}

////////////////////////////////////////////////////////////////////////
CSSM_DATA_PTR MSHN_SignData(CSSM_CSP_HANDLE hCSP,
                         CSSM_DATA_PTR pClear,  // bits to sign
                         CSSM_KEY_PTR pKey,
                         char * password)
{

   // create signature context
   CSSM_RETURN cssmstatus;
   CSSM_CC_HANDLE hSigContext;

   // this is the return value
   CSSM_DATA_PTR pSigned = (CSSM_DATA_PTR)mem_fx.malloc_func
      (sizeof(CSSM_DATA),NULL);
   pSigned->Data = NULL;
   pSigned->Length = 0;

   CSSM_CRYPTO_DATA cspData;
   CSSM_DATA       paramData;
   // Set up the crypto data
```

```
cspData.Callback = NULL;

// The "cspData" is the password for the signer's private key
cout << " password length " << strlen(password) << endl;

if (strlen(password))
        {
              paramData.Length = strlen(password);
              paramData.Data = (uint8*)
                    mem_fx.malloc_func(paramData.Length,NULL);
              memcpy(paramData.Data, password, paramData.Length);
        } else {
              paramData.Length = 0;
              paramData.Data = NULL;
        }
cspData.Param = &paramData;

cout << "sig context password length: " << paramData.Length << endl;

hSigContext = CSSM_CSP_CreateSignatureContext(hCSP,
              CSSM_ALGID_SHA1WithDSA,
              &cspData,
              pKey);

CSSM_ERROR_PTR pError = CSSM_GetError();

cout << " error creating signing context code " << pError->error
     << endl;

cout << " sig context: " << hSigContext << endl;

if (hSigContext == 0)
   {
        cout << "Error creating signature context" << endl;
        return NULL;
   }

CSSM_QUERY_SIZE_DATA queryData;

queryData.SizeInputBlock = pClear->Length;
queryData.SizeOutputBlock = 0;

CSSM_QuerySize(hSigContext,
                    CSSM_TRUE,  // for encryption
                    1,                // 1 block
                    &queryData);

pError = CSSM_GetError();

cout << " error querying signing data code " << pError->error
     << endl;
```

```
   // allocate memory to hold the encrypted bits
   pSigned->Length = queryData.SizeOutputBlock;
   pSigned->Data = (unsigned char*)
      mem_fx.malloc_func(queryData.SizeOutputBlock,NULL);

   cssmstatus = CSSM_SignData(hSigContext,
                              pClear,
                              1,
                              pSigned);

   pError = CSSM_GetError();
   cout << " error signing data code " << pError->error << endl;

   CSSM_DeleteContext(hSigContext);

      if (cssmstatus != CSSM_OK)
      {
            cout << "Signature failed" << endl;
            return NULL;
      }
      else
      {
            return pSigned;
      }
}

////////////////////////////////////////////////////////////////////////
CSSM_BOOL MSHN_VerifyData(CSSM_CSP_HANDLE hCSP,
                          CSSM_DATA_PTR pSigned,    // bits to verify
                          CSSM_DATA_PTR pClear,
                          CSSM_KEY_PTR pKey)
{

   // create signature verification context
   CSSM_BOOL cssmstatus;
   CSSM_CC_HANDLE hVerifContext;

   hVerifContext = CSSM_CSP_CreateSignatureContext(hCSP,
                           CSSM_ALGID_SHA1WithDSA,
                           NULL,    // pass phrase not needed
                           pKey);

   CSSM_ERROR_PTR pError = CSSM_GetError();

   cout << " error creating signing context code " << pError->error
        << endl;

   cout << " verif context: " << hVerifContext << endl;
```

```
    if (hVerifContext == 0)
        {
                cout << "Error creating signature context" << endl;
                return NULL;
        }

    cssmstatus = CSSM_VerifyData(hVerifContext,
                                  pClear,
                                  1,
                                  pSigned);

    pError = CSSM_GetError();
    cout << " error verifying data code " << pError->error << endl;

    CSSM_DeleteContext(hVerifContext);

    if (cssmstatus != CSSM_TRUE)
        {
                cout << "Verification failed" << endl;
                return(CSSM_FALSE);
        }
        else
        {
                cout << "\nSignature matches public key" << endl;
        return(CSSM_TRUE);
        }
}

///////////////////////////////////////////////////////////////////////////
CSSM_DATA_PTR MSHN_DigestData(CSSM_CC_HANDLE hCSP,
                              CSSM_DATA_PTR pClear)
{

    CSSM_RETURN cssmstatus;
    CSSM_CC_HANDLE hdigestContext;

    // this is the return value
    CSSM_DATA_PTR pdigest = (CSSM_DATA_PTR)mem_fx.malloc_func
        (sizeof(CSSM_DATA),NULL);
    pdigest->Data = NULL;
    pdigest->Length = 0;

    hdigestContext = CSSM_CSP_CreateDigestContext(hCSP, CSSM_ALGID_MD5);

    if (hdigestContext == 0)
        {
                cout << "Error creating digest context" << endl;
                return NULL;
        }

    cssmstatus = CSSM_DigestData(hdigestContext,pClear,1,pdigest);
```

```
   CSSM_DeleteContext(hdigestContext);

   if (cssmstatus != CSSM_OK)
      {
      cout << "Digest creation failed" << endl;
            return NULL;
      }
   else
      {
      return pdigest;
      }
}

////////////////////////////////////////////////////////////////////////
CSSM_KEY_PTR MSHN_GenerateDESKey(CSSM_CSP_HANDLE hCSP,
                                  uint32 keySizeInBits)

{
      CSSM_RETURN cssmstatus;
      CSSM_CC_HANDLE hCC = NULL;

      hCC = CSSM_CSP_CreateKeyGenContext(hCSP,          // CSP handle
                              CSSM_ALGID_DES,    // alg ID
                              NULL,       // pass phrase not req for DES
                              keySizeInBits,           // key size
                              NULL,              // seed
                              NULL,              // salt
                              NULL,       // start date
                              NULL,       // end date
                              NULL);      // params

      if(hCC == NULL)
      {
            cout << "Error creating key generation context" << endl;
            return NULL;
      }
      CSSM_KEY_PTR pKey =
      (CSSM_KEY_PTR)mem_fx.malloc_func(sizeof(CSSM_KEY),NULL);

      // setting these fields to NULL will tell the CSP to allocate the
      // memory for us
      pKey->KeyData.Data =  NULL;
      pKey->KeyData.Length = 0;

      cssmstatus = CSSM_GenerateKey(hCC,  // context handle
                              CSSM_KEYUSE_ANY, // usage
                              CSSM_KEYATTR_RETURN_DEFAULT, // attributes
                              NULL, // label
                              pKey); // the key
```

```
    CSSM_DeleteContext(hCC);

    if(cssmstatus != CSSM_OK)
    {
          cout << "Error creating CSSM key" << endl;
          return NULL;
    }
    return pKey;
}

////////////////////////////////////////////////////////////////////////////
CSSM_WRAP_KEY_PTR MSHN_WrapKey(CSSM_CSP_HANDLE hCSP,
                              CSSM_KEY_PTR psymmetricKey,
                              CSSM_KEY_PTR pmasterKey)

{
    CSSM_RETURN cssmstatus;
    CSSM_CC_HANDLE hCC = NULL;

    hCC = CSSM_CSP_CreateSymmetricContext(hCSP,  // CSP handle
                                    CSSM_ALGID_DES,    // alg ID
                                    CSSM_ALGMODE_CBCPadIV8,
                                    pmasterKey,
                                    NULL,    // no initial vector
                                    CSSM_PADDING_PKCS5,      //padding
                                    0        // 0 rounds
                                    );

    if(hCC == NULL)
    {
          cout << "Error creating symmetric context" << endl;
          return NULL;
    }
    CSSM_WRAP_KEY_PTR pKey =
          (CSSM_KEY_PTR)mem_fx.malloc_func(sizeof(CSSM_KEY),NULL);

    // setting these fields to NULL will tell the CSP to allocate the
    // memory for us
    pKey->KeyData.Data =  NULL;
    pKey->KeyData.Length = 0;

    cssmstatus = CSSM_WrapKey(hCC,           // context handle
                        NULL,                // passphrase
                        psymmetricKey,       // wrapped key
                        pKey);               // unwrapped key

    CSSM_DeleteContext(hCC);

    if(cssmstatus != CSSM_OK)
    {
          cout << "Error wrapping key" << endl;
```

```
            return NULL;
        }
        return pKey;
}

////////////////////////////////////////////////////////////////////////
CSSM_KEY_PTR MSHN_UnwrapKey(CSSM_CSP_HANDLE hCSP,
                            CSSM_WRAP_KEY_PTR psymmetricKey,
                            CSSM_KEY_PTR pmasterKey)

{
        CSSM_RETURN cssmstatus;
        CSSM_CC_HANDLE hCC = NULL;

        hCC = CSSM_CSP_CreateSymmetricContext(hCSP,   // CSP handle
                                CSSM_ALGID_DES,    // alg ID
                                CSSM_ALGMODE_CBCPadIV8,
                                pmasterKey,
                                NULL,    // no initial vector
                                CSSM_PADDING_PKCS5,     //padding
                                0        // 0 rounds
                                );
        if(hCC == NULL)
        {
                cout << "Error creating symmetric context" << endl;
                return NULL;
        }
        CSSM_KEY_PTR pKey =
        (CSSM_KEY_PTR)mem_fx.malloc_func(sizeof(CSSM_KEY),NULL);

        // setting these fields to NULL will tell the CSP to allocate the
        // memory for us
        pKey->KeyData.Data =  NULL;
        pKey->KeyData.Length = 0;

        cssmstatus = CSSM_UnwrapKey(hCC,         // context handle
                        NULL,           // passphrase
                        psymmetricKey, // wrapped key
                        CSSM_KEYATTR_RETURN_DEFAULT, // key attributes
                        NULL,           // key label
                        pKey);          // unwrapped key

        CSSM_DeleteContext(hCC);

        if(cssmstatus != CSSM_OK)
        {
                cout << "Error unwrapping key" << endl;
                return NULL;
        }
        return pKey;
}
```

L. SHOWUTIL.H

```
// Written by David Shifflett
//
#include <iostream.h>
#include <stdlib.h>
#include <cssm.h>
#include <oidscert.h>
#include <oidsalg.h>

#ifndef _showutil_h
#define _showutil_h

void * certmgr_malloc (uint32 size, void *allocRef);

void certmgr_free (void *mem_ptr, void *allocRef);

void * certmgr_realloc (void *ptr, uint32 size, void *allocRef);

void * certmgr_calloc (uint32 num, uint32 size, void *allocRef);

void fix_key_size (CSSM_KEY_PTR ppublicKey);

CSSM_BOOL match_field(CSSM_DATA_PTR pstring, const char* toMatch,
                      const char* delimiter, int field_pos);

char** Split(int* numStrings, const char* toSplit, const char*
             delimiter);

void show_error (char *err_label);

void show_pointer (uint8 *in_data, int length, char *label);

void show_data (const CSSM_DATA in_data, char *label);

void show_data_char (const CSSM_DATA in_data, char *label);

void show_data_ptr (const CSSM_DATA_PTR in_ptr, char *label, uint32
                    num_datas);

void show_data_char_ptr (const CSSM_DATA_PTR in_ptr, char *label, uint32
                    num_datas);
CSSM_BOOL isequaloid (CSSM_OID oid1, CSSM_OID oid2);

CSSM_BOOL translate_oid (CSSM_OID in_oid, char *label);

void show_oid (CSSM_OID in_oid, char *label);

void show_oid_ptr (CSSM_OID_PTR in_ptr, char *label, uint32 num_oids);
```

```
CSSM_DATA_PTR get_cert_field(CSSM_DATA_PTR certdata, CSSM_CL_HANDLE
                             clhand, CSSM_OID oid);

void show_cert_field(CSSM_DATA_PTR certdata, CSSM_CL_HANDLE clhand,
                     char *label, CSSM_OID oid, CSSM_BOOL translate);

void show_cert_fields(CSSM_DATA_PTR certdata, CSSM_CL_HANDLE clhand);

void show_ds_cert_subj (CSSM_DL_DB_HANDLE dbhand, CSSM_MODULE_HANDLE
                        clhand);

void show_ds_certs (CSSM_DL_DB_HANDLE dbhand, CSSM_MODULE_HANDLE
                    clhand);

CSSM_DATA_PTR get_subject_cert (CSSM_DL_DB_HANDLE dbhand,
                                CSSM_MODULE_HANDLE clhand,
                                char *subjectname);

void show_date (CSSM_DATE_PTR in_ptr, char *label);

void show_key (CSSM_KEY_PTR in_ptr, char *label);

void show_field_ptr (CSSM_FIELD_PTR in_ptr, char *label, uint32
                     num_fields);

void show_cl_services (void * in_ptr, uint32 num_subsvc);

void show_datastore_names (CSSM_DLSUBSERVICE_PTR sub_ptr);

void show_dl_services (void * in_ptr, uint32 num_subsvc);

void show_hard_csp (CSSM_HARDWARE_CSPSUBSERVICE_INFO subsvc);

void show_soft_csp (CSSM_SOFTWARE_CSPSUBSERVICE_INFO subsvc);

void show_csp_services (void * in_ptr, uint32 num_subsvc);

void show_tp_services (void * in_ptr, uint32 num_subsvc);

void show_services (CSSM_SERVICE_INFO_PTR svc_ptr);

void show_mod (CSSM_MODULE_INFO_PTR mod_ptr);

void show_list (CSSM_LIST_PTR csp_list, CSSM_SERVICE_MASK svcmask,
                CSSM_INFO_LEVEL info_lvl);

void show_cssm_info();

void show_all_modules();

#endif // _showutil_h
```

M. SHOWUTIL.CPP

```
// Written by David Shifflett
#include <iostream.h>
#include <stdlib.h>
#include <cssm.h>
#include <oidscert.h>
#include <oidsalg.h>
#include <x509defs.h>
#include "showutil.h"
#include "mshn_mem.h"

#ifdef DEBUG_MSHN_KEY
   char debug_in[80];
#endif

//=====================================================================
void fix_key_size (CSSM_KEY_PTR ppublicKey)
{
   switch(ppublicKey->KeyHeader.EffectiveKeySizeInBits)
      {
      case 584:
      case 528:
      case 536:
            ppublicKey->KeyHeader.EffectiveKeySizeInBits = 512;
            break;
      case 840:
      case 784:
            ppublicKey->KeyHeader.EffectiveKeySizeInBits = 768;
            break;
      case 1112:
      case 1048:
      case 1032:
            ppublicKey->KeyHeader.EffectiveKeySizeInBits = 1024;
      break;
      }
#ifdef DEBUG_MSHN_KEY
   show_key(ppublicKey, "Fix Key After");
   cout << endl << "Press enter "; cin.getline(debug_in, 80);
#endif

}

//=====================================================================
CSSM_BOOL match_field(CSSM_DATA_PTR pstring, const char* toMatch,
                      const char* delimiter, int field_pos)
{
      CSSM_BOOL retval = CSSM_FALSE;
```

```
    char** stringArray = NULL;
    int numStrings = 0;

    if (pstring && (pstring->Data) && (pstring->Length > 0) &&
       (pstring->Data[pstring->Length-1] == 0))
    {
       stringArray = Split(&numStrings, (char*)pstring->Data,
                                 delimiter);
       if (numStrings >= field_pos+1)
          {
             char *target = stringArray[field_pos];
          if (strcmp(target, toMatch) == 0){
             retval = CSSM_TRUE;
          }
       }

       // Clean up
       if (numStrings)
          {
                for (int j=0; j<numStrings; j++)
                      mshn_free(stringArray[j],NULL);
                mshn_free(stringArray,NULL);
          }
    }
  return retval;
}

//======================================================================
char** Split(int* numStrings, const char* toSplit,
             const char* delimiter)
{
    char** toReturn = NULL;
    char* pCurrent = (char *)toSplit;
    char* pNext = NULL;
    int count = 0;

    if (!toSplit || !numStrings || !delimiter) return NULL;

    // First, we have to get find the number of items
    *numStrings = 1;
    while (pCurrent)
    {
          pNext = strstr(pCurrent, delimiter);
          if (pNext) {
                (*numStrings)++;
                pCurrent = pNext + strlen(delimiter);
          } else {
                pCurrent = NULL;
          }
    }
```

```
    // Now, allocate the array of strings to return
    toReturn = (char**)mshn_malloc(sizeof(char*) * *numStrings,NULL);
    if (!toReturn) return NULL;

    // Now fill in each string
    count = 0;
    pCurrent = (char *)toSplit;
    pNext = NULL;
    while (count < *numStrings)
    {
          pNext = strstr(pCurrent, delimiter);
          if (pNext) {
             toReturn[count] = (char*)mshn_malloc(sizeof(char) *
                ((pNext - pCurrent) + 1),NULL);
             memcpy(toReturn[count], pCurrent, (pNext-pCurrent));
             toReturn[count][pNext-pCurrent] = '\0';
          } else {
                toReturn[count] = (char*)mshn_malloc(sizeof(char) *
                   (strlen(pCurrent) + 1),NULL);
                strcpy(toReturn[count], pCurrent);
          }

          // Skip past the delimiter
          pCurrent = pNext + strlen(delimiter);
          count++;
    }
    return toReturn;
}

//==================================================================
void show_error (char *err_label)
{
   CSSM_ERROR_PTR the_error = CSSM_GetError();
   if ((the_error != NULL) && (the_error->error != CSSM_OK)) {
      char *err_out = (((err_label != NULL) &&
                      (strlen(err_label) > 0)) ? err_label: "");
      cout << "Error number is (" << the_error->error
           << ") " << err_out << endl;
   }
}

//==================================================================
void show_pointer (uint8 *in_data, int length, char *label)
{
   if (in_data != NULL) {
      cout << label << " DATA is (" << length << ")" << endl;
      int line_cntr = 0;
      for (int idx=0; ((idx<length) && (line_cntr<5)); idx++) {
         cout << " " << (int)in_data[idx];
         if ((idx>0) && ((idx % 19) == 0)) {
            cout << endl;
```

```
            line_cntr++;
         }
      }
      cout << endl;
   }
}

//=====================================================================
void show_data (const CSSM_DATA in_data, char *label)
{
      if (in_data.Data != NULL) {
      cout << label << " DATA is (" << in_data.Length << ")" << endl;
      int line_cntr = 0;
      for (int idx=0; ((idx<in_data.Length) && (line_cntr<5)); idx++) {
            cout << " " << (int)in_data.Data[idx];
                  if ((idx>0) && ((idx % 19) == 0)) {
            cout << endl;
            line_cntr++;
         }
      }
      cout << endl;
   }
}

//=====================================================================
void show_data_char (const CSSM_DATA in_data, char *label)
{
      if (in_data.Data != NULL) {
      cout << label << " DATA is (" << in_data.Length << ")" << endl;
      for (int idx=0; idx<in_data.Length; idx++) {
            cout << (char)in_data.Data[idx];
      }
      cout << endl;
   }
}

//=====================================================================
void show_data_ptr (const CSSM_DATA_PTR in_ptr, char *label,
                    uint32 num_datas)
{
      cout << "Number of " << label << " DATA " << num_datas << endl;
      if ((num_datas > 0) && (in_ptr != NULL)) {
      for (int jdx=0; jdx<num_datas; jdx++) {
            show_data(in_ptr[jdx], label);
      }
   }
}

//=====================================================================
void show_data_char_ptr (const CSSM_DATA_PTR in_ptr, char *label,
                         uint32 num_datas)
```

```
{
      cout << "Number of " << label << " DATA " << num_datas << endl;
      if ((num_datas > 0) && (in_ptr != NULL)) {
      for (int jdx=0; jdx<num_datas; jdx++) {
            show_data_char(in_ptr[jdx], label);
      }
   }
}

//=====================================================================
CSSM_BOOL isequaloid (CSSM_OID oid1, CSSM_OID oid2)
{
   CSSM_BOOL result = CSSM_FALSE;
   if (oid1.Length == oid2.Length) {
      if (memcmp(oid1.Data, oid2.Data, oid1.Length) == 0) {
            result = CSSM_TRUE;
      }
   }
   return result;
}

//=====================================================================
CSSM_BOOL translate_oid (CSSM_OID in_oid, char *label)
{
      CSSM_BOOL result = CSSM_FALSE;
      if (in_oid.Data != NULL) {
            result = CSSM_TRUE;
      char *the_type;
      if (isequaloid(in_oid,
         CSSMOID_X509V3SignedCertificate) == CSSM_TRUE) {
            the_type = "CSSMOID_X509V3SignedCertificate";
            }else if (isequaloid(in_oid,
                  CSSMOID_X509V3Certificate) == CSSM_TRUE) {
                        the_type = "CSSMOID_X509V3Certificate";
            }else if (isequaloid(in_oid,
                  CSSMOID_X509V1Version) == CSSM_TRUE) {
                        the_type = "CSSMOID_X509V1Version";
            }else if (isequaloid(in_oid,
                  CSSMOID_X509V1SerialNumber) == CSSM_TRUE) {
                        the_type = "CSSMOID_X509V1SerialNumber";
            }else if (isequaloid(in_oid,
                  CSSMOID_X509V1IssuerName) == CSSM_TRUE) {
                        the_type = "CSSMOID_X509V1IssuerName";
            }else if (isequaloid(in_oid,
                  CSSMOID_X509V1ValidityNotBefore) == CSSM_TRUE) {
                        the_type = "CSSMOID_X509V1ValidityNotBefore";
            }else if (isequaloid(in_oid,
                  CSSMOID_X509V1ValidityNotAfter) == CSSM_TRUE) {
                        the_type = "CSSMOID_X509V1ValidityNotAfter";
            }else if (isequaloid(in_oid,
                  CSSMOID_X509V1SubjectName) == CSSM_TRUE) {
```

```
            the_type = "CSSMOID_X509V1SubjectName";
}else if (isequaloid(in_oid,
        CSSMOID_CSSMKeyStruct) == CSSM_TRUE) {
            the_type = "CSSMOID_CSSMKeyStruct";
}else if (isequaloid(in_oid,
        CSSMOID_X509V1SubjectPublicKeyAlgorithm)
        == CSSM_TRUE) {
        the_type = "CSSMOID_X509V1SubjectPublicKeyAlgorithm";
}else if (isequaloid(in_oid,
        CSSMOID_X509V1SubjectPublicKeyAlgorithmParameters)
        == CSSM_TRUE) {
        the_type =
        "CSSMOID_X509V1SubjectPublicKeyAlgorithmParameters";
}else if (isequaloid(in_oid,
        CSSMOID_X509V1SubjectPublicKey) == CSSM_TRUE) {
        the_type = "CSSMOID_X509V1SubjectPublicKey";
}else if (isequaloid(in_oid,
        CSSMOID_X509V1CertificateIssuerUniqueId)
        == CSSM_TRUE) {
        the_type = "CSSMOID_X509V1CertificateIssuerUniqueId";
}else if (isequaloid(in_oid,
        CSSMOID_X509V1CertificateSubjectUniqueId)
        == CSSM_TRUE) {
        the_type = "CSSMOID_X509V1CertificateSubjectUniqueId";
}else if (isequaloid(in_oid,
        CSSMOID_X509V3CertificateExtensionStruct)
        == CSSM_TRUE) {
        the_type = "CSSMOID_X509V3CertificateExtensionStruct";
}else if (isequaloid(in_oid,
        CSSMOID_X509V3CertificateNumberOfExtensions)
        == CSSM_TRUE) {
        the_type =
        "CSSMOID_X509V3CertificateNumberOfExtensions";
}else if (isequaloid(in_oid,
        CSSMOID_X509V3CertificateExtensionId) == CSSM_TRUE) {
        the_type = "CSSMOID_X509V3CertificateExtensionId";
}else if (isequaloid(in_oid,
        CSSMOID_X509V3CertificateExtensionCritical)
        == CSSM_TRUE) {
        the_type =
        "CSSMOID_X509V3CertificateExtensionCritical";
}else if (isequaloid(in_oid,
        CSSMOID_X509V3CertificateExtensionType)
        == CSSM_TRUE) {
        the_type = "CSSMOID_X509V3CertificateExtensionType";
}else if (isequaloid(in_oid,
        CSSMOID_X509V3CertificateExtensionValue)
        == CSSM_TRUE) {
        the_type = "CSSMOID_X509V3CertificateExtensionValue";
}else if (isequaloid(in_oid,
        CSSMOID_X509V1SignatureStruct) == CSSM_TRUE) {
```

```
                the_type = "CSSMOID_X509V1SignatureStruct";
            }else if (isequaloid(in_oid,
                CSSMOID_X509V1SignatureAlgorithm) == CSSM_TRUE) {
                the_type = "CSSMOID_X509V1SignatureAlgorithm";
            }else if (isequaloid(in_oid,
                CSSMOID_X509V1SignatureAlgorithmParameters)
                == CSSM_TRUE) {
                the_type =
                "CSSMOID_X509V1SignatureAlgorithmParameters";
            }else if (isequaloid(in_oid,
                CSSMOID_X509V1Signature) == CSSM_TRUE) {
                the_type = "CSSMOID_X509V1Signature";
        }else{
            result = CSSM_FALSE;
        }
        if (result == CSSM_TRUE) {
            cout << label << " OID is '" << the_type << "'" << endl;
        }
    }
    return result;
}

//=================================================================
void show_oid (CSSM_OID in_oid, char *label)
{
        if (in_oid.Data != NULL) {
        cout << label << " OID is (" << in_oid.Length << ") chars";
        for (int idx=0; idx<in_oid.Length; idx++) {
            cout << " " << (int)in_oid.Data[idx];
        }
        cout << endl;
    }
}

//=================================================================
void show_oid_ptr (CSSM_OID_PTR in_ptr, char *label, uint32 num_oids)
{
        cout << "Number of " << label << " OIDs " << num_oids << endl;
        if ((num_oids > 0) && (in_ptr != NULL)) {
        for (int jdx=0; jdx<num_oids; jdx++) {
            if (translate_oid(in_ptr[jdx], label) != CSSM_TRUE) {
                show_oid(in_ptr[jdx], label);
            }
        }
    }
}

//=================================================================
CSSM_DATA_PTR get_cert_field(CSSM_DATA_PTR certdata, CSSM_CL_HANDLE
                              clhand, CSSM_OID oid)
{
```

```
    CSSM_HANDLE ResultsHandle = NULL;
    uint32 numFields = 0;
    CSSM_DATA_PTR pData = NULL;

    if(clhand == CSSM_INVALID_HANDLE)
    {
          cout << "Invalid CL handle" << endl;
    }else if (!certdata)
    {
          cout << "No certificate data" << endl;
    }else{

    CSSM_ClearError();
          pData = CSSM_CL_CertGetFirstFieldValue(clhand,
                      certdata, &oid, &ResultsHandle, &numFields);
          if (!pData)
          {
          show_error("CSSM_CL_CertGetFirstFieldValue");
          cout << "No field found for OID" << endl;
          show_oid(oid, "cert field");
    }
  }
  return pData;
}

//======================================================================
void show_cert_field(CSSM_DATA_PTR certdata, CSSM_CL_HANDLE clhand,
                     char *label, CSSM_OID oid, CSSM_BOOL translate)
{
    CSSM_HANDLE ResultsHandle = NULL;
    uint32 numFields = 0;
    CSSM_DATA_PTR pData = NULL;

    if(clhand == CSSM_INVALID_HANDLE)
    {
          cout << "Invalid CL handle" << endl;
    }else if (!certdata)
    {
          cout << "No certificate data" << endl;
    }else{

    CSSM_ClearError();
          pData = CSSM_CL_CertGetFirstFieldValue(clhand,
                      certdata, &oid, &ResultsHandle, &numFields);

          if (!pData)
          {
             show_error("CSSM_CL_CertGetFirstFieldValue");
             cout << "No field found for OID" << endl;
             show_oid(oid, "cert field");
          }else{
```

```
                for (uint32 idx=0; idx<numFields; idx++) {
                   if (translate == CSSM_TRUE) {
                         CSSM_DATA_PTR pstring;
                   pstring = (CSSM_DATA_PTR)CSSM_CL_PassThrough (
                   clhand, 0,
                   INTEL_X509V3_PASSTHROUGH_TRANSLATE_DERNAME_TO_STRING,
                   pData);

                   show_data_char(*pstring, label);
                   mshn_free(pstring->Data, NULL);
                   }else{
                      show_data_char(*pData, label);
                   }
                mshn_free(pData->Data, NULL);
                pData = CSSM_CL_CertGetNextFieldValue(clhand,
                         ResultsHandle);
             }
                   // Finished retrieving fields
                   if (ResultsHandle)
                         CSSM_CL_CertAbortQuery(clhand, ResultsHandle);
      }
   }
}

//===================================================================
void show_cert_fields(CSSM_DATA_PTR certdata, CSSM_CL_HANDLE clhand)
{
   CSSM_KEY_PTR kData = NULL;

      if(clhand == CSSM_INVALID_HANDLE)
      {
            cout << "Invalid CL handle" << endl;
      }else if (!certdata)
      {
            cout << "No certificate data" << endl;
      }else{
      show_cert_field(certdata, clhand, "subject name",
            CSSMOID_X509V1SubjectName, CSSM_TRUE);
      show_cert_field(certdata, clhand, "issuer name",
            CSSMOID_X509V1IssuerName, CSSM_TRUE);
      show_cert_field(certdata, clhand, "serial number",
            CSSMOID_X509V1SerialNumber, CSSM_FALSE);
      show_cert_field(certdata, clhand, "valid from",
            CSSMOID_X509V1ValidityNotBefore, CSSM_FALSE);
      show_cert_field(certdata, clhand, "valid to",
            CSSMOID_X509V1ValidityNotAfter, CSSM_FALSE);
      kData = CSSM_CL_CertGetKeyInfo(clhand, certdata);
      show_key(kData, "Public Key");
      mshn_free(kData->KeyData.Data, NULL);
     }
}
```

```
//==================================================================
void show_ds_cert_subj (CSSM_DL_DB_HANDLE dbhand,
                        CSSM_MODULE_HANDLE clhand)
{
   CSSM_HANDLE ResultsHandle = NULL;
   CSSM_QUERY Query;
   CSSM_BOOL EODS;
   CSSM_DATA Data;
   CSSM_DB_UNIQUE_RECORD_PTR record_ptr;

   // Use a NULL filter to CSSM to get all certificates in database
   Query.NumSelectionPredicates = 0;
   Query.SelectionPredicate = NULL;
   Query.RecordType = CSSM_DL_DB_RECORD_CERT;
   Query.Conjunctive = CSSM_DB_NONE;
   record_ptr = CSSM_DL_DataGetFirst (dbhand, &Query, &ResultsHandle,
                  &EODS, NULL, &Data);
   show_error("CSSM_DL_DataGetFirst");
   CSSM_ClearError();

   // if end of data store before we even begin...
   if ((EODS == CSSM_TRUE) || (record_ptr == NULL)) {
      cout << "Couldn't get first record" << endl;
   }

   int cntr = 1;
   while ((EODS == CSSM_FALSE))
      {
         // now show some of the certificate fields
         show_cert_field(&Data, clhand, "subject name",
            CSSMOID_X509V1SubjectName, CSSM_TRUE);
         CSSM_DL_FreeUniqueRecord(dbhand,record_ptr);

         // Get the next certificate
         record_ptr = CSSM_DL_DataGetNext (dbhand, ResultsHandle, &EODS,
             NULL, &Data);
         show_error("CSSM_DL_DataGetNext");
         CSSM_ClearError();
      }

      // Done querying for information
      if (ResultsHandle)
            CSSM_DL_AbortQuery(dbhand, ResultsHandle);
}

//==================================================================
void show_ds_certs (CSSM_DL_DB_HANDLE dbhand, CSSM_MODULE_HANDLE clhand)
{
   CSSM_HANDLE ResultsHandle = NULL;
   CSSM_QUERY Query;
   CSSM_BOOL EODS;
```

```
   CSSM_DATA Data;
   CSSM_DB_UNIQUE_RECORD_PTR record_ptr;

   // Use a NULL filter to CSSM to get all certificates in database
   Query.NumSelectionPredicates = 0;
   Query.SelectionPredicate = NULL;
   Query.RecordType = CSSM_DL_DB_RECORD_CERT;
   Query.Conjunctive = CSSM_DB_NONE;
   record_ptr = CSSM_DL_DataGetFirst (dbhand, &Query, &ResultsHandle,
       &EODS, NULL, &Data);
   show_error("CSSM_DL_DataGetFirst");
   CSSM_ClearError();

   // if end of data store before we even begin...
   if ((EODS == CSSM_TRUE) || (record_ptr == NULL)) {
      cout << "Couldn't get first record" << endl;
   }

   int cntr = 1;
   while ((EODS == CSSM_FALSE))
      {
         cout << "Data record number " << cntr++ << endl;
         // show_data(Data, "from data store");

         // now show some of the certificate fields
         show_cert_fields(&Data, clhand);
         CSSM_DL_FreeUniqueRecord(dbhand,record_ptr);

         // Get the next certificate
         record_ptr = CSSM_DL_DataGetNext (dbhand, ResultsHandle, &EODS,
             NULL, &Data);
         show_error("CSSM_DL_DataGetNext");
         CSSM_ClearError();
      }

      // Done querying for information
      if (ResultsHandle)
            CSSM_DL_AbortQuery(dbhand, ResultsHandle);
}

//=====================================================================
CSSM_DATA_PTR get_subject_cert (CSSM_DL_DB_HANDLE dbhand,
                                CSSM_MODULE_HANDLE clhand,
                                char *subjectname)
{
   CSSM_DATA_PTR retval = NULL;
   CSSM_HANDLE ResultsHandle = NULL;
   CSSM_QUERY Query;
   CSSM_BOOL EODS;
   CSSM_DATA_PTR certdata =
   (CSSM_DATA_PTR)mshn_malloc(sizeof(CSSM_DATA_PTR), NULL);
```

```
    CSSM_DB_UNIQUE_RECORD_PTR record_ptr;

    // Use a NULL filter to CSSM to get all certificates in database
    Query.NumSelectionPredicates = 0;
    Query.SelectionPredicate = NULL;
    Query.RecordType = CSSM_DL_DB_RECORD_CERT;
    Query.Conjunctive = CSSM_DB_NONE;
    record_ptr = CSSM_DL_DataGetFirst (dbhand, &Query, &ResultsHandle,
        &EODS, NULL, certdata);
    show_error("CSSM_DL_DataGetFirst");
    CSSM_ClearError();

    // if end of data store before we even begin...
    if ((EODS == CSSM_TRUE) || (record_ptr == NULL)) {
       cout << "Couldn't get first record" << endl;
    }else{
       int cntr = 1;
       int found = 0;
       while ((EODS == CSSM_FALSE) && !found)
          {
          // now find the certificate matching the subject
          CSSM_DATA_PTR sdata = get_cert_field(certdata, clhand,
                                CSSMOID_X509V1SubjectName);
          CSSM_DATA_PTR pstring;
          pstring = (CSSM_DATA_PTR)CSSM_CL_PassThrough (clhand, 0,
                    INTEL_X509V3_PASSTHROUGH_TRANSLATE_DERNAME_TO_STRING,
                    sdata);

          show_data_char(*pstring, "trying subject name");
          CSSM_DL_FreeUniqueRecord(dbhand,record_ptr);

          if (match_field(pstring, subjectname, ";", 4) == CSSM_TRUE) {
             found = 1;
             retval = certdata;
          }else{
             // Get the next certificate
             record_ptr = CSSM_DL_DataGetNext (dbhand, ResultsHandle,
                    &EODS, NULL, certdata);
             show_error("CSSM_DL_DataGetNext");
             CSSM_ClearError();
          }
       }
    }

       // Done querying for information
    if (ResultsHandle)
             CSSM_DL_AbortQuery(dbhand, ResultsHandle);
    return retval;
}
```

```
//==================================================================
void show_date (CSSM_DATE_PTR in_ptr, char *label)
{
   if (in_ptr == NULL) {
      cout << label << " NULL date input" << endl;
   }else{
      cout << label << " year:month:day " << in_ptr->Year << ":"
           <<  in_ptr->Month << ":" << in_ptr->Day << endl;
   }
}

//==================================================================
void show_key (CSSM_KEY_PTR in_ptr, char *label)
{
   if (in_ptr == NULL) {
      cout << label << " NULL key input" << endl;
   }else{
      char *sdt = " starting date";
      char *edt = "  ending date ";
      char *slabel = (char*)mshn_malloc
            (strlen(label)+strlen(sdt)+1, NULL);
      strcpy(slabel, label);
      strcat(slabel, sdt);
      char *elabel = (char*)mshn_malloc
            (strlen(label)+strlen(edt)+1, NULL);
      strcpy(elabel, label);
      strcat(elabel, edt);
      CSSM_KEYHEADER kh = in_ptr->KeyHeader;
      cout << label << " header version " << kh.HeaderVersion << endl;
      cout << label << " blob type      " << kh.BlobType << endl;
      cout << label << " format         " << kh.Format << endl;
      cout << label << " algorithm ID   " << kh.AlgorithmId << endl;
      cout << label << " key class      " << kh.KeyClass << endl;
      cout << label << " key size, bits " << kh.EffectiveKeySizeInBits
           << endl;
      cout << label << " key attributes " << kh.KeyAttr << endl;
      cout << label << " key usage      " << kh.KeyUsage << endl;
      show_date(&(kh.StartDate), slabel);
      show_date(&(kh.EndDate), elabel);
      cout << label << " wrap alg ID    " << kh.WrapAlgorithmId << endl;
      cout << label << " wrap mode      " << kh.WrapMode << endl;
      show_data(in_ptr->KeyData, label);
      mshn_free(slabel, NULL);
      mshn_free(elabel, NULL);
   }
}

//==================================================================
void show_field_ptr (CSSM_FIELD_PTR in_ptr, char *label, uint32
                     num_fields)
{
```

```
      cout << "Number of " << label << " FIELDs " << num_fields << endl;
      if ((num_fields > 0) && (in_ptr != NULL)) {
      // for (int jdx=0; jdx<num_fields; jdx++) {
         // cout << "Not NULL" << endl;
         // CSSM_OID oid_val = in_ptr->FieldOid;
         // uint32 zxzx = (uint32)in_ptr;
         // cout << "Not NULL2" << endl;
         // cout << zxzx << endl;
         // CSSM_FIELD field_val = in_ptr[jdx];
         // show_oid(field_val.FieldOid, label);
         // show_data(field_val.FieldValue, label);
      // }
   }
}

//==================================================================
void show_cl_services (void * in_ptr, uint32 num_subsvc)
{
      if (in_ptr != NULL) {
      CSSM_CLSUBSERVICE_PTR sub_ptr = (CSSM_CLSUBSERVICE_PTR) in_ptr;
      for (int idx=0; idx<num_subsvc; idx++) {
         cout << "Sub-service ID " << sub_ptr->SubServiceId << endl;
         cout << "Description " << sub_ptr->Description << endl;
         cout << "Cert type " << sub_ptr->CertType << endl;
         cout << "Cert encoding " << sub_ptr->CertEncoding << endl;
         cout << "Authentication Mechanism "
              << sub_ptr->AuthenticationMechanism << endl;
         show_oid_ptr(sub_ptr->CertTemplates, "Template",
            sub_ptr->NumberOfTemplateFields);
         cout << "Num translate types "
              << sub_ptr->NumberOfTranslationTypes << endl;
         cout << "Num encoder prods. "
              << sub_ptr->WrappedProduct.NumberOfEncoderProducts
              << endl;
         cout << "Num CAs accessible "
              << sub_ptr->WrappedProduct.NumberOfCAProducts << endl;
         sub_ptr++;
      }
   }else{
      cout << "NULL subservice" << endl;
   }
}

//==================================================================
void show_datastore_names (CSSM_DLSUBSERVICE_PTR sub_ptr)
{
      if (sub_ptr != NULL) {
      if (sub_ptr->DataStoreNames != NULL)
      {
         for (int idx1=0; idx1 < sub_ptr->DataStoreNames->NumStrings;
              idx1++) {
```

```
                cout << sub_ptr->DataStoreNames->String[idx1] << endl;
            }
        }
    }
}

//===================================================================
void show_dl_services (void * in_ptr, uint32 num_subsvc)
{
      if (in_ptr != NULL) {
      CSSM_DLSUBSERVICE_PTR sub_ptr = (CSSM_DLSUBSERVICE_PTR) in_ptr;
      for (int idx=0; idx<num_subsvc; idx++) {
            cout << "Sub-service ID " << sub_ptr->SubServiceId << endl;
            cout << "Description " << sub_ptr->Description << endl;
            cout << "Type " << sub_ptr->Type << endl;
            if (sub_ptr->Type == CSSM_DL_ODBC) {
                  cout << "OBDC attributes" << endl;
            }else{
                  cout << "Unknown attribute type" << endl;
            }
            cout << "Num rel. oper. "
                 << sub_ptr->NumberOfRelOperatorTypes << endl;
            cout << "Num conj. oper. "
                 << sub_ptr->NumberOfConjOperatorTypes << endl;
            cout << "Query limits supported ";
            if(sub_ptr->QueryLimitsSupported == CSSM_TRUE) {
                  cout << "TRUE" << endl;
            }else{
               cout << "FALSE" << endl;
            }
            cout << "Num data stores " << sub_ptr->NumberOfDataStores
                 << endl;
            if (sub_ptr->DataStoreNames != NULL)
            {
               cout << "There are "
                    << sub_ptr->DataStoreNames->NumStrings
                    << " names" << endl;
               show_datastore_names(sub_ptr);
            }else{
                  cout << "Data Store Names is NULL" << endl;
            }
         sub_ptr++;
      }
   }else{
      cout << "NULL subservice" << endl;
   }
}

//===================================================================
void show_hard_csp (CSSM_HARDWARE_CSPSUBSERVICE_INFO subsvc)
{
```

```
      cout << "HARDWARE subservice" << endl;
}

//==================================================================
void show_soft_csp (CSSM_SOFTWARE_CSPSUBSERVICE_INFO subsvc)
{
      cout << "SOFTWARE subservice" << endl;
      cout << "Number of capabilities "
           << subsvc.NumberOfCapabilities << endl;
      // show_context(subsvc.CapabilityList,
      // "SOFTWARE CSP capabilities",
      // subsvc.NumberOfCapabilities);
      // if (in_ptr != NULL) {
   // }else{
   // }
}

//==================================================================
void show_csp_services (void * in_ptr, uint32 num_subsvc)
{
      if (in_ptr != NULL) {
      CSSM_CSPSUBSERVICE_PTR sub_ptr = (CSSM_CSPSUBSERVICE_PTR) in_ptr;
      for (int idx=0; idx<num_subsvc; idx++) {
            cout << "Sub-service ID " << sub_ptr->SubServiceId << endl;
            cout << "Description " << sub_ptr->Description << endl;
            cout << "Flags " << sub_ptr->CspFlags << endl;
            cout << "Custom Flags " << sub_ptr->CspCustomFlags << endl;
            cout << "Access Flags " << sub_ptr->AccessFlags << endl;
            cout << "CSP type " << sub_ptr->CspType << endl;
         if (sub_ptr->CspType == CSSM_CSP_SOFTWARE) {
            show_soft_csp(sub_ptr->SoftwareCspSubService);
         }else{
            show_hard_csp(sub_ptr->HardwareCspSubService);
         }
         sub_ptr++;
      }
   }else{
      cout << "NULL subservice" << endl;
   }
}

//==================================================================
void show_tp_services (void * in_ptr, uint32 num_subsvc)
{
      if (in_ptr != NULL) {
      CSSM_TPSUBSERVICE_PTR sub_ptr = (CSSM_TPSUBSERVICE_PTR) in_ptr;
      for (int idx=0; idx<num_subsvc; idx++) {
         cout << "Sub-service ID " << sub_ptr->SubServiceId << endl;
         cout << "Description " << sub_ptr->Description << endl;
         cout << "Cert type " << sub_ptr->CertType << endl;
         cout << "Authentication Mechanism "
```

```
                << sub_ptr->AuthenticationMechanism << endl;
            show_field_ptr(sub_ptr->PolicyIdentifiers, "Policy Id",
                           sub_ptr->NumberOfPolicyIdentifiers);
            sub_ptr++;
        }
    }else{
        cout << "NULL subservice" << endl;
    }
}

//=======================================================================
void show_services (CSSM_SERVICE_INFO_PTR svc_ptr)
{
        if (svc_ptr != NULL) {
        cout << "Description " << svc_ptr->Description << endl;
        cout << "Type " << svc_ptr->Type << endl;
        cout << "Flags " << svc_ptr->Flags << endl;
        cout << "Number Of Sub-Services "
             << svc_ptr->NumberOfSubServices << endl;
        if (svc_ptr->Type == CSSM_SERVICE_CL) {
              show_cl_services(svc_ptr->SubServiceList,
                  svc_ptr->NumberOfSubServices);
        }else if (svc_ptr->Type == CSSM_SERVICE_DL) {
              show_dl_services(svc_ptr->SubServiceList,
                  svc_ptr->NumberOfSubServices);
        }else if (svc_ptr->Type == CSSM_SERVICE_CSP) {
              show_csp_services(svc_ptr->SubServiceList,
                  svc_ptr->NumberOfSubServices);
        }else if (svc_ptr->Type == CSSM_SERVICE_TP) {
              show_tp_services(svc_ptr->SubServiceList,
                     svc_ptr->NumberOfSubServices);
        }
    }
}

//=======================================================================
void show_mod (CSSM_MODULE_INFO_PTR mod_ptr)
{
        if (mod_ptr != NULL) {
        cout << "Version (Major) " << mod_ptr->Version.Major << endl;
        cout << "Description " << mod_ptr->Description << endl;
        cout << "Vendor " << mod_ptr->Vendor << endl;
        cout << "Flags " << mod_ptr->Flags << endl;
        cout << "Service Mask " << mod_ptr->ServiceMask << endl;
        cout << "Number Of Services " << mod_ptr->NumberOfServices
             << endl;
        show_services(mod_ptr->ServiceList);
    }
}
```

```
//=====================================================================
void show_list (CSSM_LIST_PTR csp_list, CSSM_SERVICE_MASK svcmask,
                                CSSM_INFO_LEVEL info_lvl)
{
      if (csp_list != NULL) {
      cout << endl << "Number of modules " << csp_list->NumberItems
           << endl;
      CSSM_LIST_ITEM_PTR csp_ptr = csp_list->Items;
      for (int idx=0; idx<csp_list->NumberItems; idx++) {
         cout << csp_ptr[idx].GUID.Data1 <<" "<< csp_ptr[idx].Name
              << endl;
         // Now get the module info
         CSSM_ClearError();
         CSSM_MODULE_INFO_PTR mod_ptr = CSSM_GetModuleInfo
                  (&(csp_ptr[idx].GUID),
                   svcmask, CSSM_ALL_SUBSERVICES,
                   info_lvl);
         if (mod_ptr != NULL) {
            show_mod(mod_ptr);
         }else{
            show_error("CSSM_GetModuleInfo");
         }
         CSSM_FreeModuleInfo(mod_ptr);
      //    csp_ptr++;
      }
   }else{
      cout << "Empty list" << endl;
      }
}

//=====================================================================
void show_cssm_info()
{
   // now get the cssm info
   CSSM_CSSMINFO_PTR cssm_info_ptr;
   CSSM_ClearError();
   cssm_info_ptr = CSSM_GetInfo();
   if (cssm_info_ptr != NULL) {
      cout << "CSSM description    " << cssm_info_ptr->Description
           << endl;
      cout << "CSSM vendor         " << cssm_info_ptr->Vendor << endl;
      cout << "CSSM location       " << cssm_info_ptr->Location << endl;
      cout << "CSSM version(major) " << cssm_info_ptr->Version.Major
           << endl;
      cout << "CSSM version(minor) " << cssm_info_ptr->Version.Minor
           << endl;
      if (cssm_info_ptr->ThreadSafe == CSSM_TRUE) {
            cout << "CSSM is thread safe" << endl;
      }else{
            cout << "CSSM NOT thread safe" << endl;
      }
```

```
   }else{
      show_error("CSSM_GetInfo");
   }
   CSSM_FreeInfo(cssm_info_ptr);
   CSSM_ClearError();
   CSSM_RETURN verify = CSSM_VerifyComponents();
   if (verify == CSSM_FALSE) {
      show_error("CSSM_VerifyComponents");
   }
   CSSM_ClearError();
}

//=====================================================================
void show_all_modules()
{
   // now get the list of modules
   CSSM_LIST_PTR csp_list;
   csp_list = CSSM_ListModules(CSSM_SERVICE_CSP, CSSM_FALSE);
   show_list(csp_list, CSSM_SERVICE_CSP, CSSM_INFO_LEVEL_SUBSERVICE);
   CSSM_FreeList(csp_list);
   csp_list = CSSM_ListModules(CSSM_SERVICE_DL, CSSM_FALSE);
   show_list(csp_list, CSSM_SERVICE_DL, CSSM_INFO_LEVEL_ALL_ATTR);
   CSSM_FreeList(csp_list);
   csp_list = CSSM_ListModules(CSSM_SERVICE_CL, CSSM_FALSE);
   show_list(csp_list, CSSM_SERVICE_CL, CSSM_INFO_LEVEL_ALL_ATTR);
   CSSM_FreeList(csp_list);
   csp_list = CSSM_ListModules(CSSM_SERVICE_TP, CSSM_FALSE);
   show_list(csp_list, CSSM_SERVICE_TP, CSSM_INFO_LEVEL_SUBSERVICE);
   CSSM_FreeList(csp_list);
}
```

N. MSHN_DEMO.H

```
//******************************************************
//******************************************************
// File:  mshn_demo.h
// Name:  David Shifflett
//
// Project: MSHN
//
// Operating Environment: Windows 95/Windows NT
// Compiler: Borland C++ for Windows
// Date:  18 MAY 98
//
// Description: MSHN demonstration constant definitions
//******************************************************

#ifndef _MSHN_DEMO_H
#define _MSHN_DEMO_H
```

```
// MSHN demonstration constant definitions
// Ports to be used for demonstration communications
const int PORT_CLIENT_SCHEDULER     = 5001;
const int PORT_CLIENT_RESOURCE      = 5002;
const int PORT_RESOURCE_RSS         = 5003;
const int PORT_RESOURCE_RRD         = 5004;

// IP addresses of demonstration computers
const char *IP_CLIENT         = "131.120.10.89";
const char *IP_SCHEDULER      = "131.120.10.90";
const char *IP_RSS            = "131.120.10.90";
const char *IP_RRD            = "131.120.10.90";
const char *IP_RESOURCE_1     = "131.120.10.94";

// Filename for shared key storage
const char *key_fname         = "mshn.key";

// Filename for shared 'REQUEST_DB'
const char *reqdb_fname       = "mshn.rdb";

#endif
```

O. MSHN_TYPES.H

```
//******************************************************
//******************************************************
// File:  mshn_types.h
// Name:  David Shifflett
//
// Project: MSHN
//
// Operating Environment: Windows 95/Windows NT
// Compiler: Borland C++ for Windows
// Date:  18 MAY 98
//
// Description: MSHN demonstration type definitions
//******************************************************

#ifndef _MSHN_TYPES_H
#define _MSHN_TYPES_H

// MSHN demonstration type definitions
// Data passing structure, to be used for keys, certificates,
// chunks of data to be encrypted, decrypted, signed, etc.
typedef struct mshn_data {
  unsigned int    the_length;
  unsigned char *the_data;
} mshn_data;
```

```
// Communication security options
typedef enum comm_security {
   COMSEC_NONE = 0,           // No communications security
   COMSEC_INT  = 1,           // Communications are signed
   COMSEC_CON  = 2,           // Communications are encrypted
   COMSEC_BOTH = 3,           // Communications are signed and encrypted
} comm_security;

// Certificate validation levels
typedef enum cert_checking {
   CERT_NONE = 0,       // No certificate validation
   CERT_AUTH = 1,       // Check certificate authenticity
   CERT_REV  = 2,       // Check for certificate revocation
   CERT_BOTH = 3        // Check certificate authenticity and revocation
} cert_checking;

// Signed fields bitmasks
#define SIGN_NONE       0x0         // No fields are signed
#define SIGN_ALL        0x1         // All fields are signed
#define SIGN_1          0x2         // First field is signed
#define SIGN_2          0x4         // Second field is signed
#define SIGN_3          0x8         // Third field is signed
#define SIGN_4          0x10        // Fourth field is signed
#define SIGN_5          0x20        // Fifth field is signed
#define SIGN_6          0x40        // Sixth field is signed
#define SIGN_7          0x80        // Seventh field is signed

// MSHN signature
typedef struct mshn_sig {
  int  fields_set; // Use 'signed fields bitmasks' to set/check
  mshn_data signature;   // The actual signature
} mshn_sig;

// MSHN security token
typedef struct mshn_token {
  int   request_id;           // Job request identifier
  mshn_data job_session_key;  // Job session key, should be encrypted
} mshn_token;

#endif
```

P. MSHN_DEFS.H

```
//*****************************************************
// File:  mshn_defs.h
// Name:  David Shifflett
//
// Project: MSHN
//
```

```
// Operating Environment: Windows 95/Windows NT
// Compiler: Borland C++ for Windows
// Date: 18 MAY 98
//
// Description: MSHN constant definitions
//*****************************************************

#ifndef _MSHN_DEFS_H
#define _MSHN_DEFS_H

// MSHN booleans
#define MSHN_TRUE           1
#define MSHN_FALSE          0

#endif
```

Q. MSHN_MEM.H

```
// MSHN Memory Function Header File
// Written by David Shifflett
#ifndef _MSHN_MEM_H
#define _MSHN_MEM_H

typedef unsigned int uint32;

//  memory function headers
void * mshn_malloc (uint32 size, void *allocRef);

void  mshn_free (void *mem_ptr, void *allocRef);

void * mshn_realloc (void *ptr, uint32 size, void *allocRef);

void * mshn_calloc (uint32 num, uint32 size, void *allocRef);
#endif
```

R. MSHN_ERR.H

```
//*****************************************************
// File:  mshn_err.h
// Name:  David Shifflett
//
// Project: MSHN
//
// Operating Environment: Windows 95/Windows NT
// Compiler: Borland C++ for Windows
// Date: 18 MAY 98
//
// Description: MSHN error code definitions
//*****************************************************
```

```
#ifndef _MSHN_ERR_H
#define _MSHN_ERR_H

// MSHN Security layer error codes
#define MSHN_OK                        0
#define MSHN_UNKNOWN_ERROR            -1
#define MSHN_SL_BASE_ERROR            -1000
#define MSHN_NOT_INITIALIZED          (MSHN_SL_BASE_ERROR -1)
#define MSHN_INITIALIZED              (MSHN_SL_BASE_ERROR -2)
#define MSHN_ALG_NOT_FOUND            (MSHN_SL_BASE_ERROR -3)
#define MSHN_CERT_NOT_FOUND           (MSHN_SL_BASE_ERROR -4)
#define MSHN_CERT_INVALID             (MSHN_SL_BASE_ERROR -5)
#define MSHN_CERT_REVOKED             (MSHN_SL_BASE_ERROR -6)
#define MSHN_INVALID_SIG              (MSHN_SL_BASE_ERROR -7)

#endif
```

APPENDIX C. MSHN DEMONSTRATION OPERATING INSTRUCTIONS

A. EQUIPMENT/SOFTWARE SETUP

The demonstration program is configured to run on three personal computers using the Windows NT operating system. These computers must be linked via a network that supports TCP/IP communications protocol. IP addresses and port numbers can be modified to specify the particular computers you will use for the demonstration. These parameters are stored in the MSHN_DEMO.H file.

The demonstration program requires the use of Intel's CDSA version 1.2. This software may be obtained from Intel at their web site (http://developer.intel.com/ial/security). The MSHN demonstration program assumes you have installed CDSA in the default directory: "c:\cdsa_1.2\".

The MSHN demonstration program is written in C++. We used Borland C++ 5.0 to compile the source code. The MSHN demonstration described in this appendix assumes you have compiled the source code and placed the executable files in the following directory: "c:\cdsa_1.2\demo9\".

Intel's CDSA comes with a sample application program called "certmgr.exe" This program can be used to create a certificate database, certificates, and public/private key pairs. Prior to running the MSHN demonstration, you must create a certificate database for each personal computer used in the demo. The MSHN demonstration program assumes you will name the certificate database on each PC "mshn". You must create a certificate for each user and compute resource along with the corresponding public/private key pairs. You must also create a MSHN core certificate and distribute it to the core components you will be using. The certmgr program has a utility for importing and exporting certificates to floppy disk. Use this utility to distribute the MSHN core certificate to each of the PC's serving as a MSHN core component.

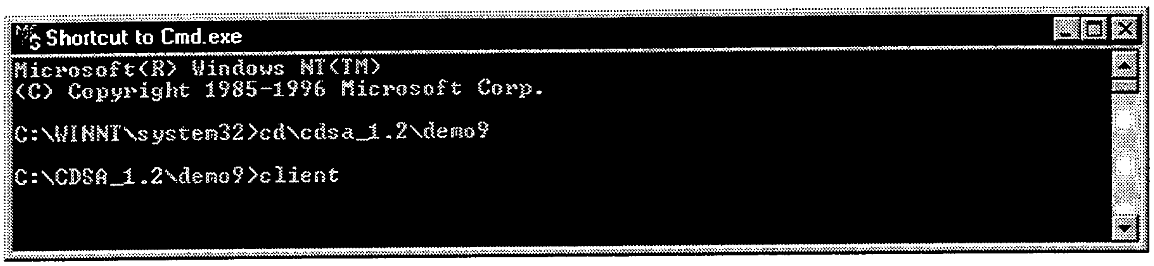
Intel's CDSA does not come with a public/private key encryption algorithm. Therefore, a symmetric (DES) key is used in its place. This shared symmetric key must be created and distributed to each of the components before execution. The demonstration assumes you will store this key in a file called "mshn.key".

B. EXECUTING THE DEMO

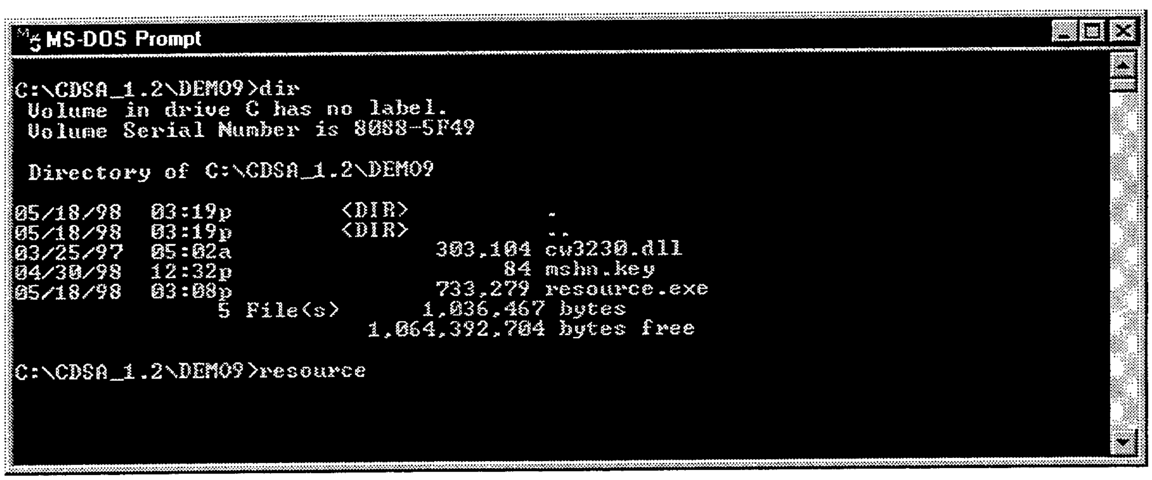
1. Start Up

Follow these instructions for operating the MSHN demonstration program. Instructions are followed by an illustration. The illustrations show the program output as the demonstration progresses.

On the machine that will serve as the client, open an MS-DOS window and execute the command "client".



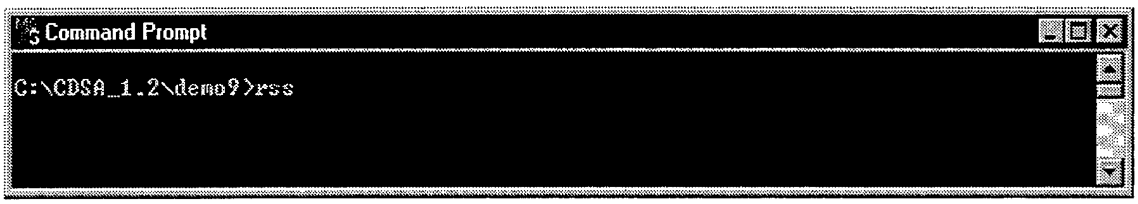
On the machine that will serve as the resource, open an MS-DOS window and execute the command "resource".



On the machine that will serve as the MSHN core, you must open three MS-DOS Windows.

In the first window, execute the program "scheduler".

In the second window, execute the program "rss".



In the third window execute the program "rrd".

Starting the scheduler: After you start the scheduler program, the scheduler will ask you to enter your certificate name.

Enter the name of the MSHN core certificate. For our demonstration, the certificate is named "MSHN core".

Next you must enter the pass phrase for the private key of the MSHN core certificate. Note that the pass phrase will not echo to the display.

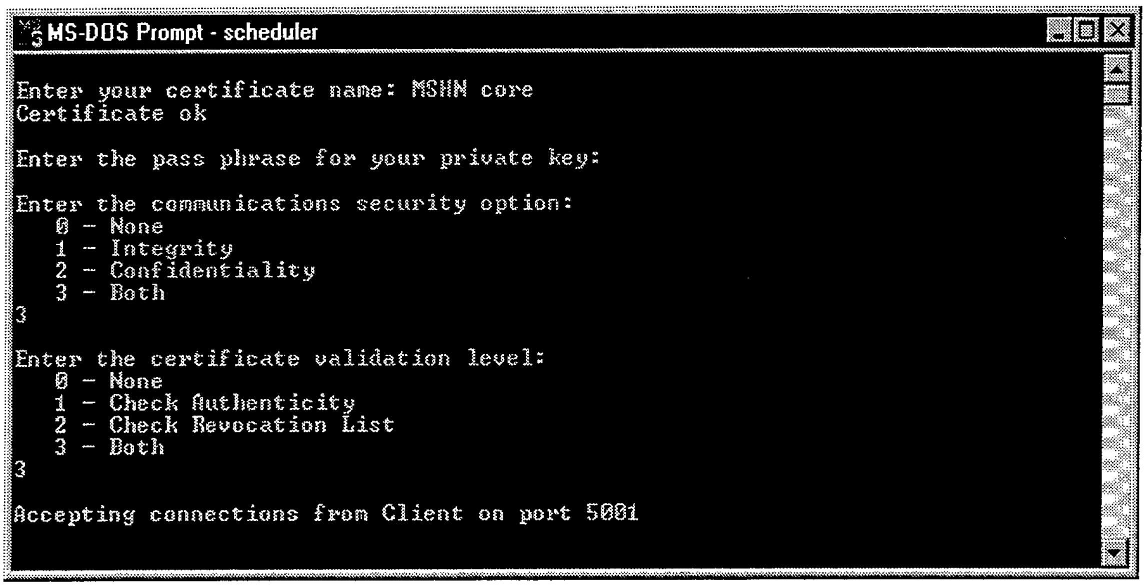
The pass phrase is: cdsacdsa.

Next the scheduler will ask you to enter the required communications security option.

You may choose from none, integrity, confidentiality or both.

Next the scheduler will ask you to enter the required certificate validation level.

You may choose from none, check authenticity, check revocation list, or both.



Now the scheduler is ready to accept connections from the client.

Starting the resource status server: After you start the resource status server program, it will ask you to enter the certificate name for the MSHN core.

For our demonstration, the certificate is named "MSHN core".

Next you must enter the pass phrase for the private key of the MSHN core certificate.

The pass phrase is: cdsacdsa.

Next the RSS will ask you to enter the required communications security option.

You may choose from none, integrity, confidentiality or both.

Next the RSS will ask you to enter the required certificate validation level.

You may choose from none, check authenticity, check revocation list, or both.

Command Prompt - rss

```
Enter your certificate name: MSHN core
Certificate ok

Enter the pass phrase for your private key:

Enter the communications security option:
   0 - None
   1 - Integrity
   2 - Confidentiality
   3 - Both
3

Enter the certificate validation level:
   0 - None
   1 - Check Authenticity
   2 - Check Revocation List
   3 - Both
3

Accepting connections from Resource on port 5003
```

Now the RSS is ready to accept connections from the compute resource.

Starting the resource requirements database: After you start the resource requirements database program, it will ask you to enter the certificate name for the MSHN core.

For our demonstration, the certificate is named "MSHN core".

Next you must enter the pass phrase for the private key of the MSHN core certificate.

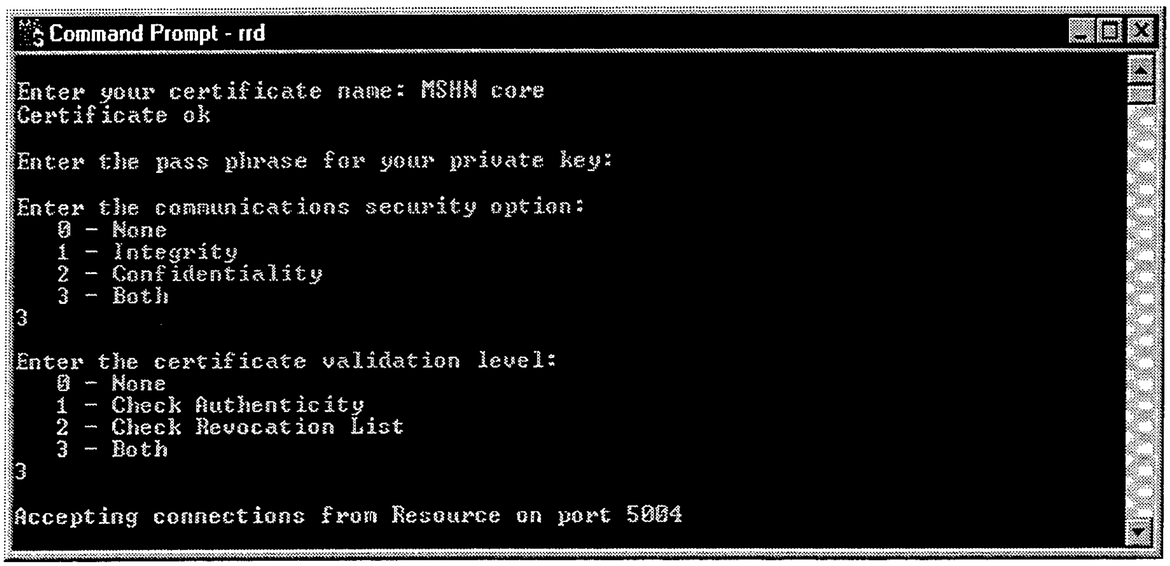
The pass phrase is: cdsacdsa.

Next the RRD will ask you to enter the required communications security option.

You may choose from none, integrity, confidentiality or both.

Next the RRD will ask you to enter the required certificate validation level.

You may choose from none, check authenticity, check revocation list, or both.



Now the RRD is ready to accept connections from the compute resource.

Starting the compute resource: After you start the compute resource program, it will ask you to enter the certificate name for the compute resource. For our demonstration, the certificate is named "resource1".

Next you must enter the pass phrase for the private key of the resource certificate.

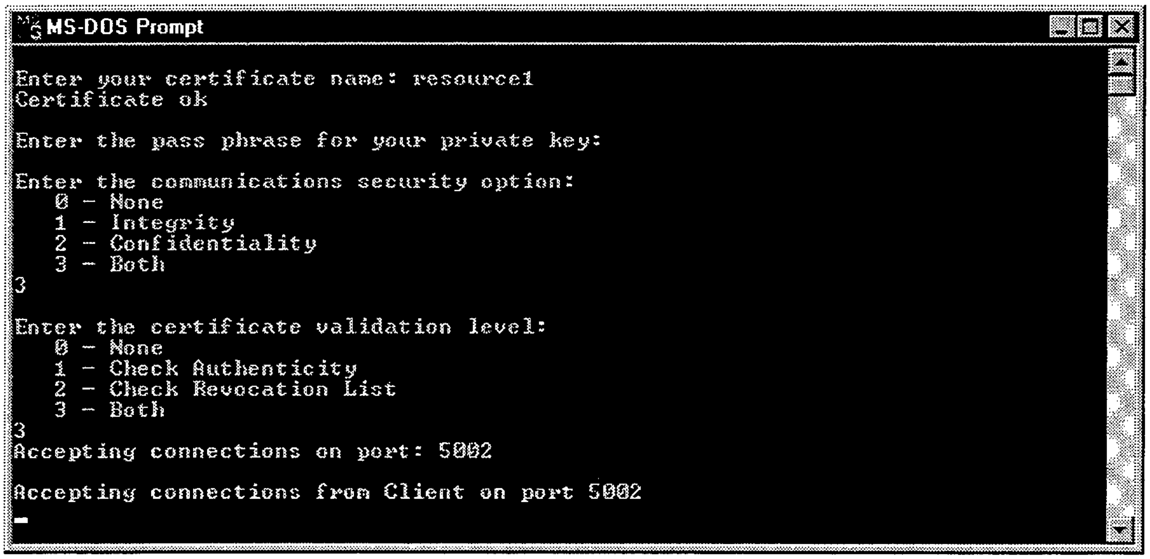
The pass phrase is: cdsacdsa.

Next the resource will ask you to enter the required communications security option.

You may choose from none, integrity, confidentiality or both.

Next the resource will ask you to enter the required certificate validation level.

You may choose from none, check authenticity, check revocation list, or both.



Now the resource is ready to accept connections from the client.

Starting the MSHN client: After you start to the MSHN client shell program, it will ask you to enter the certificate name for the user. For our demonstration, the certificate is named "Roger Wright".

Next you must enter the pass phrase for the private key of the user's certificate.

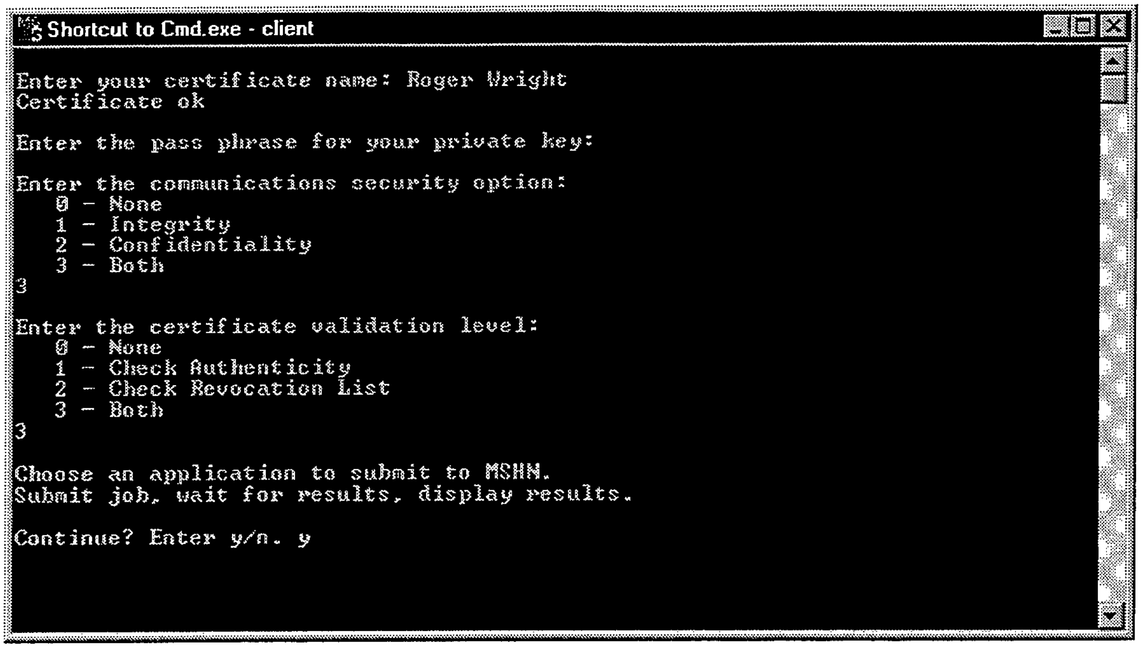
The pass phrase is: password.

Next the client will ask you to enter the required communications security option.

You may choose from none, integrity, confidentiality or both.

Next the client will ask you to enter the required certificate validation level.

You may choose from none, check authenticity, check revocation list, or both.



Now you must choose an application to run. Enter your choice from one to five.

2. Job execution.

The client has signed, encrypted, and bundled the user ID, user certificate, schedule information, and signature.

```
Connection made
Our socket is <372>.
Our port is <5001>.
Our address is <131.120.10.90>.
Our current state is <Connected>.

total length 908
Sending everything: 908 bytes.
Sent <908> bytes.
Communications Security Option          <3>
User ID                          length <16> bytes.
User Certificate                 length <800> bytes.
Schedule Info                    length <16> bytes.
Client Signature                 length <56> bytes.
Sent <908> bytes.
```

This bundle has been transmitted to the scheduler. The scheduler has received the bundle, decrypted it, and verified the signature.

The scheduler pauses to display the user name and application submitted. Make sure the scheduler window is active and press enter to continue.

```
MS-DOS Prompt - scheduler

Accepting connections from Client on port 5001
Received <908> bytes.
Communication security option          <3>
Client ID                       length <16> bytes.
Client certificate              length <800> bytes.
Job information                 length <16> bytes.
Client signature                length <56> bytes.
Received <908> bytes.


Number of User Id DATA 1
User Id DATA is <12>
Roger Wright
Number of Job Info DATA 1
Job Info DATA is <12>
Application3

Press enter _
```

The scheduler simulates the calculation of a scheduling solution. The scheduler encrypts and signs a bundle containing the compute resource ID, job information, security token, job session key, and MSHN core

certificate. This bundle is transmitted back to the client. The scheduler now waits for the next job scheduling request.

```
MS-DOS Prompt - scheduler
total length 1104
Sending everything: 1104 bytes.
Sent (1104) bytes.
Resource ID            length (16) bytes.
Job information        length (40) bytes.
Security token         length (96) bytes.
Job session key        length (88) bytes.
MSHN core certificate  length (792) bytes.
Scheduler signature    length (48) bytes.

Accepting connections from Client on port 5001
_
```

The client receives the scheduling information bundle from the scheduler. This bundle is decrypted and verified.

The client creates a signed and encrypted bundle containing the communications security option, job information, security token, and user certificate. This bundle is transmitted to the compute resource.

```
Shortcut to Cmd.exe
Received (1104) bytes.
Resource ID            length (16) bytes.
Job Info               length (40) bytes.
Security Token         length (96) bytes.
Job Session Key        length (88) bytes.
MSHN Core Certificate  length (792) bytes.
Scheduler Signature    length (48) bytes.
Sending job to compute resource...
Connection made
Our socket is (372).
Our port is (5002).
Our address is (131.120.10.94).
Our current state is (Connected).

total length 1004
Sending everything: 1004 bytes.
Sent (1004) bytes.
Communications Security Option         (3)
Job Info                        length (40) bytes.
Security Token                  length (96) bytes.
User Certificate                length (800) bytes.
Client Signature                length (48) bytes.
Sent (1004) bytes.
```

The compute resource receives the job information bundle from the client. The resource decrypts and verifies the bundle.

The resource executes the user's application and collects the results. The results are signed and encrypted. The results are then transmitted back to the client.

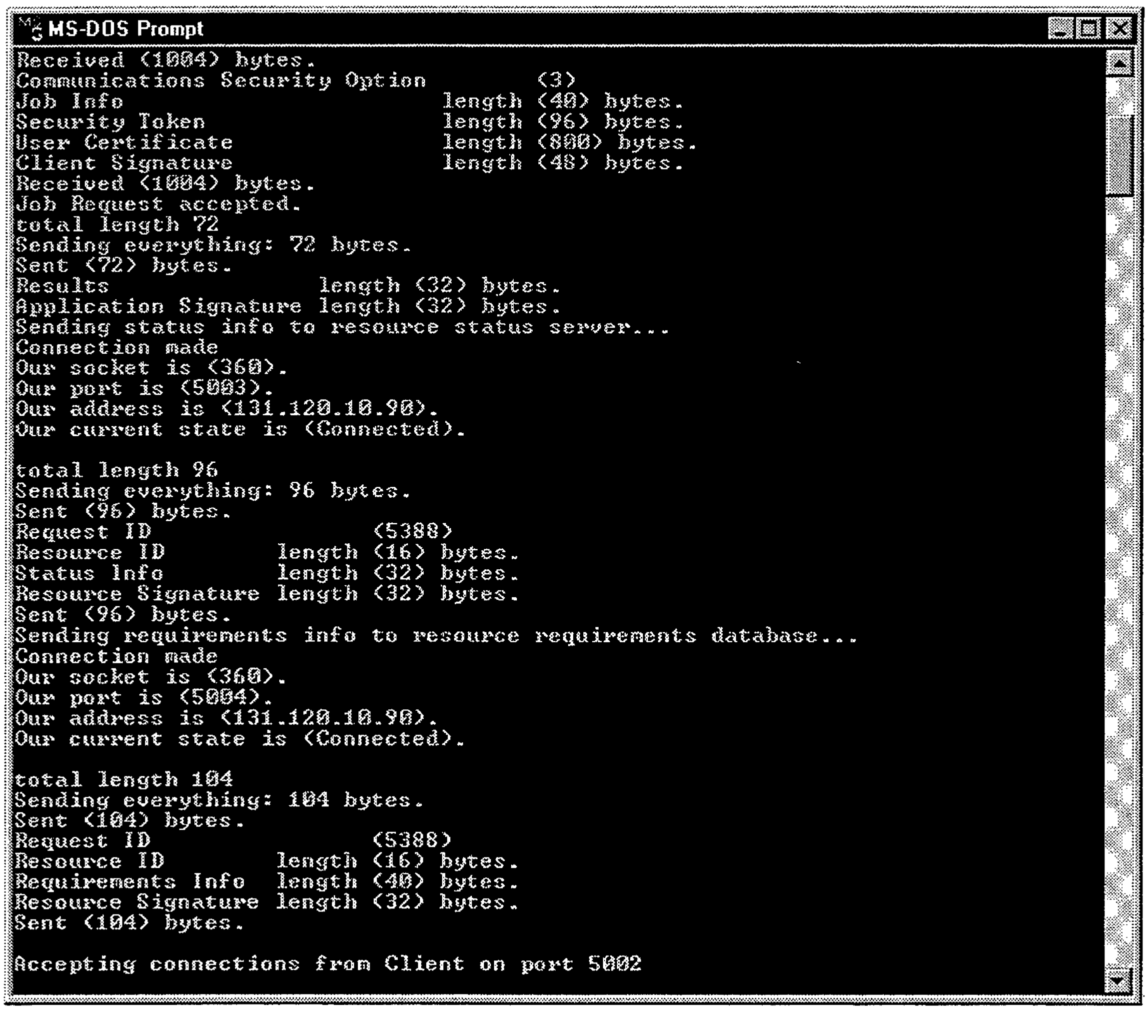
The compute resource signs and encrypts a bundle containing the resource ID and status information for the job just completed.

This bundle is transmitted to the resource status server.

The compute resource signs and encrypts a bundle containing the resource ID and requirements information for the job just completed.

This bundle is transmitted to the resource requirements database.

The compute resource has finished processing the job and now waits for the next job request.



```
MS-DOS Prompt
Received (1004) bytes.
Communications Security Option          (3)
Job Info                         length (40) bytes.
Security Token                   length (96) bytes.
User Certificate                 length (800) bytes.
Client Signature                 length (48) bytes.
Received (1004) bytes.
Job Request accepted.
total length 72
Sending everything: 72 bytes.
Sent (72) bytes.
Results               length (32) bytes.
Application Signature length (32) bytes.
Sending status info to resource status server...
Connection made
Our socket is (360).
Our port is (5003).
Our address is (131.120.10.90).
Our current state is (Connected).

total length 96
Sending everything: 96 bytes.
Sent (96) bytes.
Request ID                  (5388)
Resource ID        length (16) bytes.
Status Info        length (32) bytes.
Resource Signature length (32) bytes.
Sent (96) bytes.
Sending requirements info to resource requirements database...
Connection made
Our socket is (360).
Our port is (5004).
Our address is (131.120.10.90).
Our current state is (Connected).

total length 104
Sending everything: 104 bytes.
Sent (104) bytes.
Request ID                  (5388)
Resource ID        length (16) bytes.
Requirements Info  length (40) bytes.
Resource Signature length (32) bytes.
Sent (104) bytes.

Accepting connections from Client on port 5002
```

The resource requirements database receives the bundle from the compute resource. The bundle is decrypted and verified. The RRD displays the

user, the resource, and the requirements information. The RRD has finished processing the job and now waits for the next update.

```
Command Prompt - rrd
Received (104) bytes.
Job request ID                       (5388)
Resource ID                   length (16) bytes.
Resource requirements info length (40) bytes.
Resource signature            length (32) bytes.
Received (104) bytes.

For request ID: 5388
FOUND: communication security: 3
FOUND: user id                : Roger Wright
FOUND: session key:
session key DATA is (84)
 2 0 0 0 51 46 90 225 134 3 209 17 156 109 0 160 201 112 129 98
 0 0 0 0 0 0 0 0 14 0 0 0 2 0 0 0 40 0 0
 0 0 0 0 0 0 0 0 128 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 15
 8 0 2 23 13 1 0
Received update from resource
Resource id                : resource1
Requirements info          : This is sample job requirements data

Accepting connections from Resource on port 5004
```

The resource status server receives the bundle from the compute resource. The bundle is decrypted and verified. The RSS displays the user, the resource, and the resources status information. The RSS has finished processing the job and now waits for the next update.

```
Command Prompt - rss
Received (96) bytes.
Job request ID                (5388)
Resource ID            length (16) bytes.
Resource status info length (32) bytes.
Resource signature     length (32) bytes.
Received (96) bytes.

For request ID: 5388
FOUND: communication security: 3
FOUND: user id                : Roger Wright
FOUND: session key:
session key DATA is (84)
 2 0 0 0 51 46 90 225 134 3 209 17 156 109 0 160 201 112 129 98
 0 0 0 0 0 0 0 0 14 0 0 0 2 0 0 0 40 0 0
 0 0 0 0 0 0 0 0 128 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 15
 8 0 2 23 13 1 0
Received update from resource
Resource id                : resource1
Resource status info       : This is sample job status data

Accepting connections from Resource on port 5003
```

The client receives the results bundle from the compute resource. This bundle is decrypted and verified.

The client displays the application results and allows the user the option to submit another job to MSHN.

```
Received <72> bytes.
Results               length <32> bytes.
Application Signature length <32> bytes.

**********   APPLICATION   RESULTS   **********
This is sample job results

Choose an application to submit to MSHN.
Submit job, wait for results, display results.

Continue? Enter y/n. n

C:\CDSA_1.2\demo9>_
```

To stop execution of the servers, press "Ctrl C".

This completes the MSHN security services demonstration program.

LIST OF REFERENCES

1. Stevens, Rick., Woodward, Paul., DeFanti, Tom., Catlett, Charlie., *From the I-WAY to the National Technology Grid*, Communications of the ACM, Vol. 40, No. 11, November, 1997.

2. Department of Defense., *Joint Vision 2010*. [Online]. Available: http://www.dtic.mil/doctrine/jv2010/jvpub.htm [1998 February 12].

3. Department of Defense., *Joint Pub 6-0: Doctrine for Command, Control, Communications, and Computer (C4) Systems Support to Joint Operations*, National Defense University Press, May 1995.

4. Case, Fred T., Hines, Christopher W. and Steven W. Satchwell, *Analysis of Air Operations During Desert Shield/Desert Storm*, US Air Force Studies and Analyses Agency, 1991.

5. Freund, Richard., and others, *Scheduling Resources in Multi-User, Heterogeneous, Computing Environments with SmartNet*, Proceedings of the Seventh Heterogeneous Computing Workshop, IEEE Computer Society, Los Alamitos, Calafornia, March 1998.

6. Kresho, John P., *Quality Network Load Information Improves Performance of Adaptive Applications*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1997.

7. Department of Defense Computer Security Center, DoD 5200.28-STD, *Department of Defense Trusted Computer System Evaluation Criteria*, December 1985.

8. Anderson, J. *Computer Security Technology Planning Study*, ESD-TR-73-51, Vol. I, AD-758206, ESD/AFSC, Hanscom AFB MA, October 1972.

9. Garfinkle S., and Spafford G., *Practical UNIX and Internet Security*, O'Reilly & Associates, Sebastopol, California 1996.

10. Stinson, Douglas R., *Cryptography: Theory and Practice*, CRC Press, Boca Raton, Florida 1995.

11. Stallings, William., *Network and Internetwork Security: Principles and Practice*, Prentice Hall, Englewood Cliffs, New Jersey 1995.

12. Intel Corporation., *Common Data Security Architecture Specification Release 1.2*. [Online]. Available: http://developer.intel.com/ial/security/ [1998 February].

13. Kaufman, C., Perlman, R., Speciner, M. *Network Security: Private Communication in a Public World,* Prentice Hall, Englewood Cliffs, New Jersey, 1995.

14. SETCo, (1997 May 31) The SET Standard Technical Specification [Online]. Available: http://www.setco.org/set_specifications.html

15. Netscape Communications, *SSL 3.0 Specification* [Online]. Available: http://www.netscape.com/libr/ssl/ssl3/.

16. Dusse, S., and others, (1998 March) *RFC2311: S/MIME Version 2 Message Specification* [Online]. Available: http://info.internet.isi.edu/in-notes/rfc/files/rfc2311.txt

17. The Open Group. (1998 January 8). *New Security Standard From The Open Group Brings the Realization of High-Value E-Commerce For Everyone a Step Further* [Online]. Available: http://www.opengroup.org/press/6jan98 [1998 February 17].

18. Foster, I., and others, *A Secure Communications Infrastructure for High-Performance Distributed Computing,* Proceedings from the 6th IEEE Symposium on Distributed Computing, 1997.

19. Wulf, W., Wang, C., Kienzle, D., *A New Model of Security for Distributed Systems*, Technical Report UVa CS Technical Report CS-95-34, University of Virginia, Richmond, Virginia, August 1995.

20. Irvine, Cynthia E., *Trust Relationships*, to appear in Encyclopedia of Distributed Computing, ed. J. Urban and P. Dasgupta, Kluwer Academic Publishers.

21. Wright, R., Shifflett, D., Irvine, C., *Security for a Virtual Heterogeneous Machine*, to appear in the 14th Computer Security Applications Conference, December 1998.

22. Shirley, Lawrence J., Schell, Roger R. *Mechanism Sufficiency Validation by Assignment*, IEEE Symposium on Security and Privacy, Oakland, California, April 1981.

23. Object Management Group, Inc., *The Common Object Request Broker Architecture and Specification*, OMG Document 93-12-43, December, 1993.

24. Object Management Group, Inc., *CORBA Security*, OMG Document 95-12-1, December, 1995.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center................2
   8725 John J. Kingman Rd., Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library.................................2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Dr. Dan Boger.......................................2
   Chairman, Code CS
   Naval Postgraduate School
   Monterey, CA 93943-5000

4. Dr. Cynthia E. Irvine...............................5
   Computer Science Department Code CS/Ic
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. David Shifflett.....................................1
   Computer Science Department Code CS
   Naval Postgraduate School
   Monterey, CA 93943-5000

6. Dr. Debra Hensgen...................................1
   Computer Science Department Code CS/Hd
   Naval Postgraduate School
   Monterey, CA 93943-5000

7. Dr. William Kemple..................................1
   Joint C4I Department Code 39
   Naval Postgraduate School
   Monterey, CA 93943-5000

8. CPT Roger E. Wright.................................2
   5923 Clear Ridge Rd.
   Elkridge, MD 21075

9. Mr. George Bieber...................................1
   Defense Information Systems Agency
   5113 Leesburg Pike, Suite 400
   Falls Church, VA 22041-3230

10. Commander..1
Naval Security Group Command
ATTN: Mr. James Shearer
Naval Security Group Headquarters
9800 Savage Road
Suite 6585
Fort G. Meade, MD 20755-6585

11. Commander..1
Space and Naval Warfare Systems Command
ATTN: CAPT Dan Galik
PMN 161
53560 Hull Street
OT-1 Room 1025
San Diego, CA 92152-5002

12. Dr. Blaine W. Burnham.............................1
R23
National Security Agency
9800 Savage Road.
Fort George G. Meade, MD 20755-600